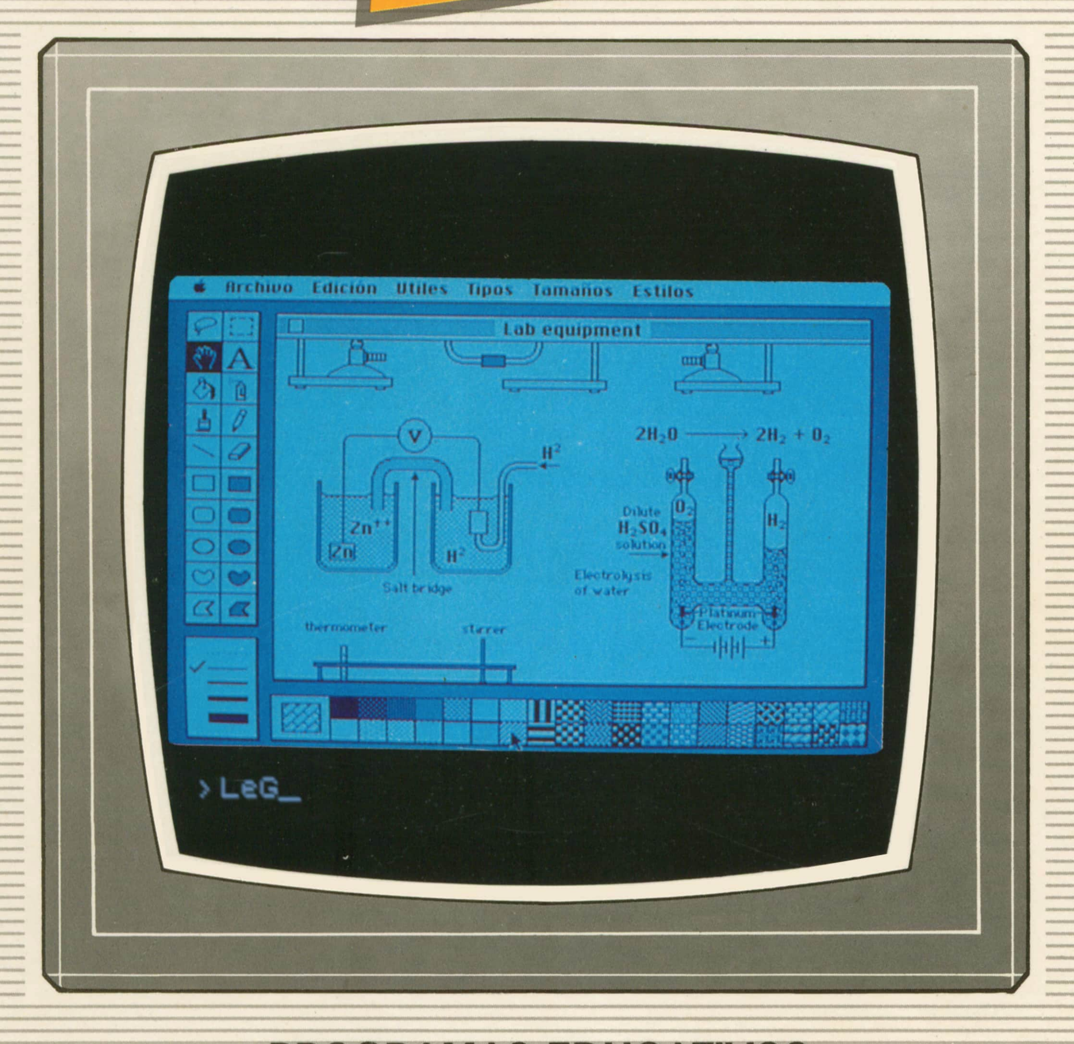
▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación

29

PASO A PASO



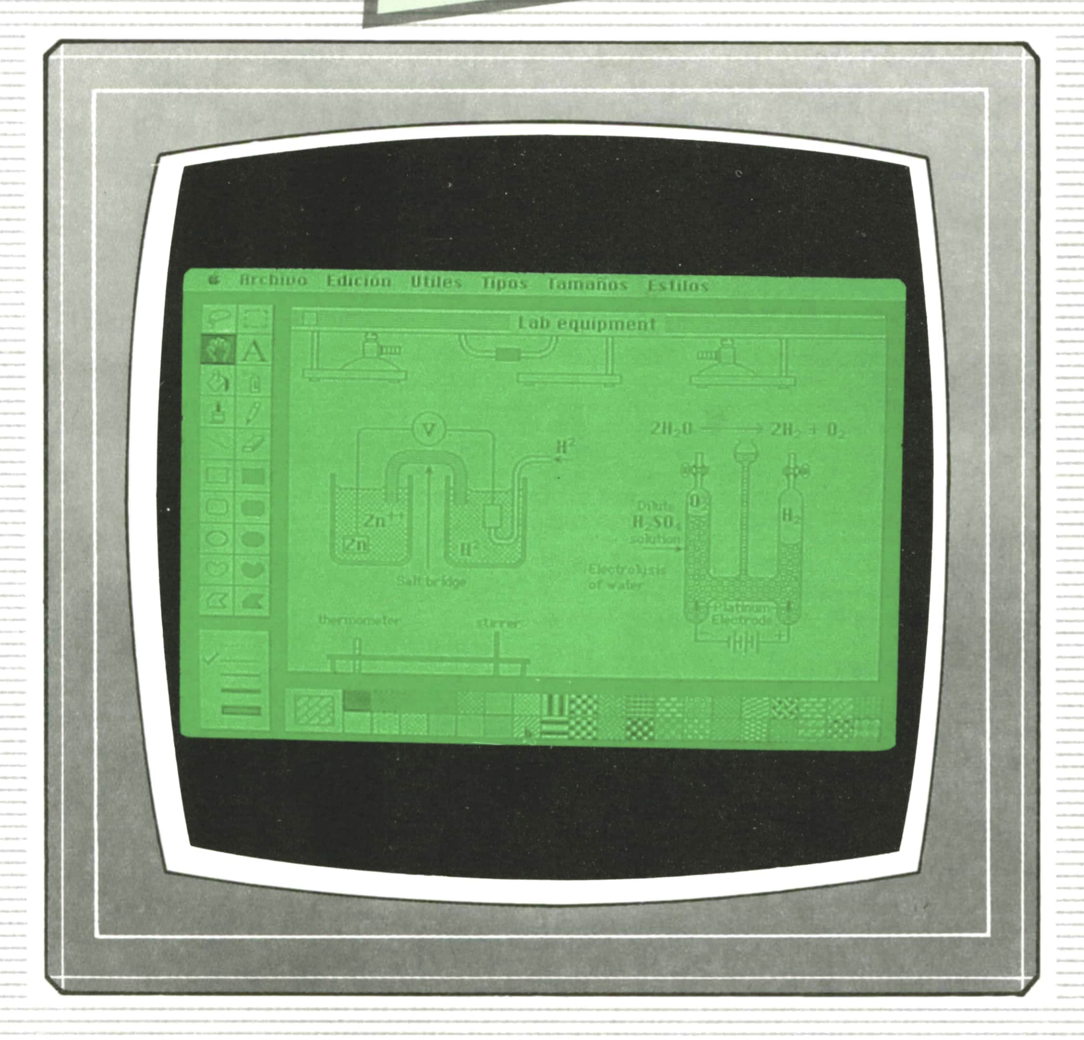
PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 29

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.
Gerente:
ANTONIO G. CUERPO.
Directora de producción:
MARIA LUISA SUAREZ PEREZ.
Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.
Diseño y maquetación:
BRAVO-LOFISH.
Fotografía:
EQUIPO GALATA.
Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Jorge Thomas, Técnico de Informática. OTROS LENGUAJES (COBOL): Joaquín Salvachúa, Diplomado en Telecomunicación. José Luis Tojo, Diplomado en Telecomunicación.

Ediciones Siglo Cultural, S.A.
Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 455 09 99. 28020 Madrid.
Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.
Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.
Distribución en Ecuador: Muñoz Hnos.
Distribución en Perú: DISELPESA.
Distribución en Chile: Alfa Ltda.
Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

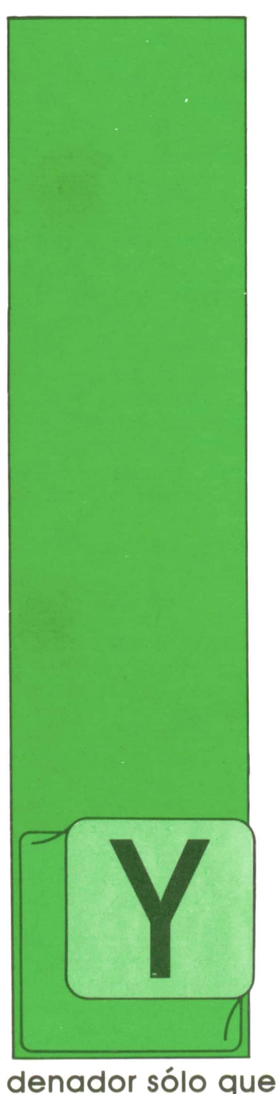
ISBN del tomo: 84-7688-176-2
ISBN de la obra: 84-7688-068-7
Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.
Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 259 73 31. 28020 Madrid.
Diciembre, 1987.
P.V.P. Canarias: 335,–.

INDICE

BASIC

MATRICES (I)



Variables con subíndice

YA hemos visto que mediante la utilización de bucles es posible introducir un gran número de datos con un solo INPUT. El problema es que en la memoria del ordenador sólo queda almacenado el último dato introducido.

Sin embargo, en muchas ocasiones es necesario que todos los datos queden almacenados en memoria y para ello tendremos que utilizar las *variables con subíndices.*

Las variables con subíndices son agrupaciones formadas por una serie de variables simples, por ejemplo:

A(1), A(2), A(3), A(4)

En este ejemplo en concreto, el nombre de la variable propiamente dicho es la A, mientras que el número encerrado entre paréntesis es el subíndice.

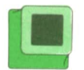
Al igual que sucede con las variables normales, existen dos tipos de variables con subíndices: numéricas (enteras, simple precisión o doble precisión) y alfanuméricas. Los nombres se forman siguiendo las mismas reglas de nomenclatura que las variables simples.

Normalmente, las variables con subíndice que tienen el mismo nombre se utilizan para almacenar datos con alguna característica común, por ejemplo, una lista de números premiados en un sorteo o una lista de alumnos de un colegio. Este conjunto de variables constituye lo que se denomina una *matriz.*

Una matriz se caracteriza por tener un nombre genérico para todo el conjunto de variables con subíndice que la integran.

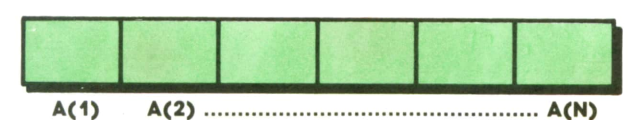
Para hacer referencia a una variable concreta del conjunto que integran la matriz es necesario dar el nombre genérico de la matriz a la que pertenece e indicar a continuación y entre paréntesis el índice o índices necesarios para localizar la variable dentro de la matriz.

Las matrices pueden ser de una, dos, tres o más dimensiones, utilizando respectivamente variables con uno, dos, tres o más subíndices, siempre encerrados entre paréntesis y separados por comas. Por tanto, vamos a analizar las matrices según el número de dimensiones.



Matrices unidimensionales

Las matrices unidimensionales se caracterizan por ser agrupaciones lineales de variables simples. En ocasiones se las denomina también listas. Podríamos representarlas esquemáticamente como se muestra en la figura 1.




Representación gráfica de una matriz unidimensional numérica.

Podemos observar que la matriz de la figura está constituida por N elementos. Los subíndices son consecutivos, por tanto, sería muy fácil introducir los datos utilizando un bucle.

Veamos un ejemplo. El programa 1 sirve para almacenar en una matriz numérica ocho números aleatorios, que bien podrían ser el resultado de un sorteo.

```
10 REM ************************
20 REM *   CARGAR UNA MATRIZ   *
30 REM ************************
40 CLS
50 FOR I=1 TO 8
60 LET P(I)=INT(RND(1)*100)+1
70 NEXT I
80 PRINT "YA ESTA CARGADA LA MATRIZ"
```

El funcionamiento es sencillo. Al ejecutarse el bucle FOR-NEXT la variable I toma el valor 1 y el primer número al azar se almacena en la variable P(1). En la segunda vuelta del bucle I toma el valor 2 y el nuevo número aleatorio generado se almacena en la variable P(2). El proceso se repite hasta que el último dato se almacena en P(8).

De aquí podemos deducir que los bucles FOR-NEXT están íntimamente relacionados con las matrices y se utilizan en multitud de operaciones relacionadas con éstas, por ejemplo, cargar la matriz, es decir, introducir los datos, imprimir los datos de la matriz, buscar un dato, etc.

El programa 1 funciona correctamente en cualquier ordenador, excepto en el SPECTRUM. Sin embargo, si en lugar de una matriz de 8 elementos manejara una matriz de 11, ya no funcionaría en ninguna máquina. Esto se debe a que siempre que vayamos a utilizar una matriz con más de 11 elementos tendremos que "avisar" previamente al ordenador. En el caso del SPECTRUM hay que avisarle siempre, independientemente del tamaño de la matriz.

El "aviso" se realiza con la instrucción DIM, cuyo formato es el siguiente:

DIM <nombre de variable> (n.° de elementos)

La instrucción DIM hace que el ordenador reserve espacio en su memoria para albergar la matriz especificada.

El formato indicado para DIM sólo es exacto para el SPECTRUM, aunque se puede utilizar para cualquier ordenador. Sin embargo, si queremos aprovechar al máximo la memoria de cualquier otra máquina es mejor utilizar el formato siguiente:

DIM <nombre de variable> (n.° de elementos – 1)

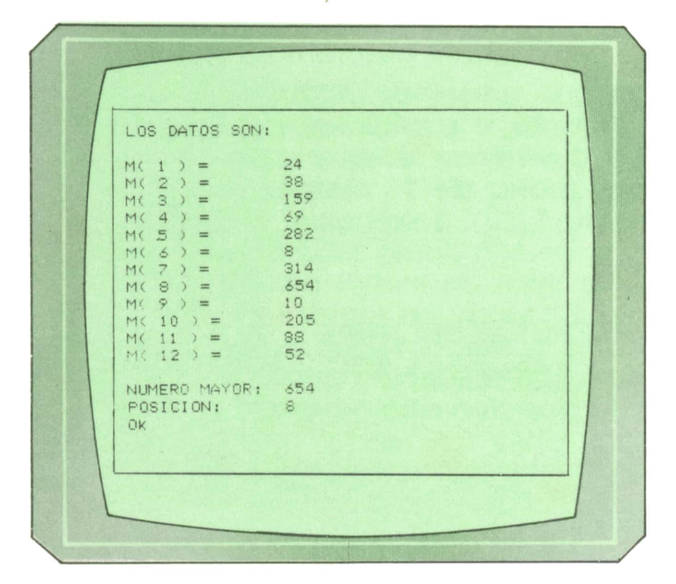
ya que podemos utilizar el subíndice 0. Por tanto, si escribimos, por ejemplo, DIM E(20) estamos reservando memoria para una matriz de 21 elementos, desde E(0) hasta E(20). Solamente el SPECTRUM no admite el 0 como subíndice.

Veamos un nuevo ejemplo. El programa 2 carga una matriz con una serie de datos numéricos introducidos por teclado. A continuación busca el dato mayor y la posición que ocupa en la lista.

```
10 REM ******************
20 REM *  NUMERO MAYOR   *
30 REM ******************
40 CLS
50 INPUT "NUMERO DE DATOS";N
60 IF N<1 THEN GOTO 50
70 CLS
80 DIM M(N)
90 FOR I=1 TO N
100 INPUT "TECLEA UN NUMERO";M(I)
110 CLS
120 NEXT I
130 LET MA=M(1)
140 LET PO=1
150 FOR I=2 TO N
160 IF M(I)>MA THEN LET MA=M(I):LET PO=I
170 NEXT I
180 PRINT "LOS DATOS SON:":PRINT
190 FOR I=1 TO N
200 PRINT "M(";I;") =",M(I)
210 NEXT I
220 PRINT
230 PRINT "NUMERO MAYOR:",MA
240 PRINT "POSICION:",PO
```

En la línea 80 se dimensiona la matriz M para que pueda almacenar N elementos. El bucle FOR-NEXT de las líneas 90 a 110 permite cargar la matriz introduciendo números desde el teclado. En las líneas 130 y 140 se considera, en principio, que el número mayor es el primero

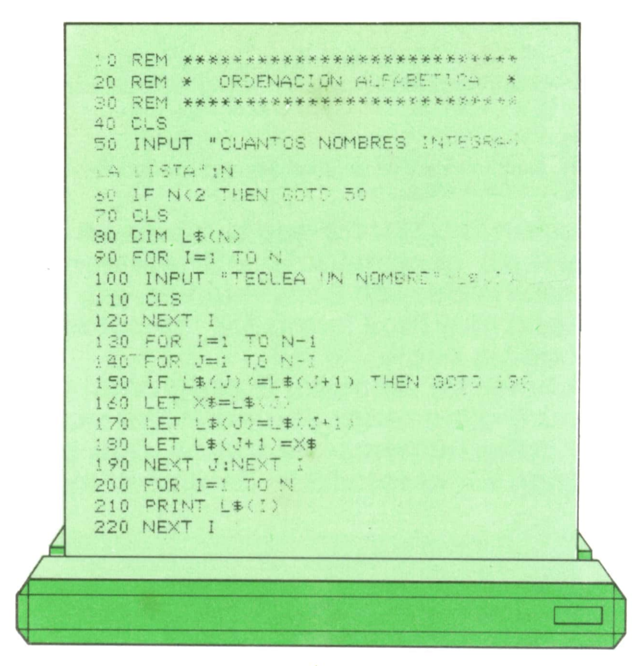
de la matriz y, por tanto, ocupa la posición 1. El bucle FOR-NEXT de las líneas 150 a 170 permite ir comparando cada elemento de la matriz con el mayor hasta el momento, de forma que si dicho elemento es mayor se almacenará en la variable MA, donde está en cada momento el número mayor de todos los examinados. En la variable PO se almacena la posición que ocupa en la matriz. Finalmente se imprimen los resultados como se muestra en la figura 2.




Presentación en pantalla del programa 2.

En cuanto a las matrices alfanuméricas tienen las mismas características que las numéricas, salvo que el nombre debe ir acompañado del signo $.

El programa 3 se encarga de ordenar alfabéticamente una lista de nombres introducidos por teclado.

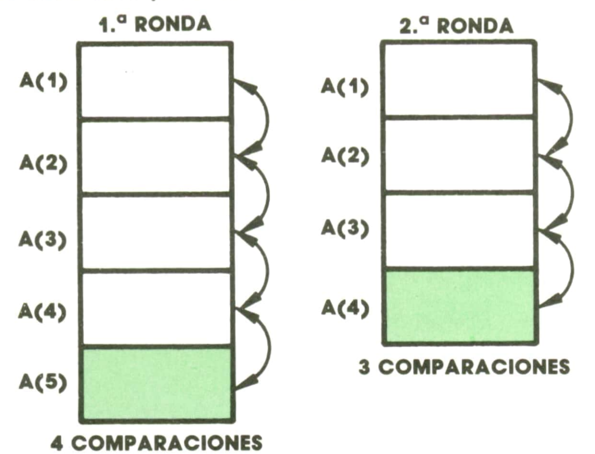


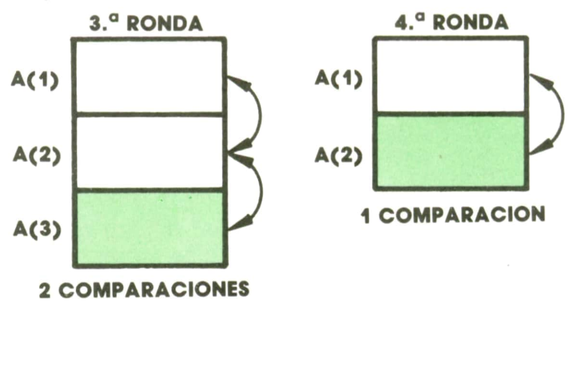
```
10 REM ****************************
20 REM *  ORDENACION ALFABETICA  *
30 REM ****************************
40 CLS
50 INPUT "CUANTOS NOMBRES INTEGRAN
LA LISTA";N
60 IF N<2 THEN GOTO 50
70 CLS
80 DIM L$(N)
90 FOR I=1 TO N
100 INPUT "TECLEA UN NOMBRE";L$(I)
110 CLS
120 NEXT I
130 FOR I=1 TO N-1
140 FOR J=1 TO N-I
150 IF L$(J)<=L$(J+1) THEN GOTO 190
160 LET X$=L$(J)
170 LET L$(J)=L$(J+1)
180 LET L$(J+1)=X$
190 NEXT J:NEXT I
200 FOR I=1 TO N
210 PRINT L$(I)
220 NEXT I
```

Para la ordenación de la lista hemos utilizado el mismo método ya visto anteriormente en esta colección, sólo que ahora más depurado.

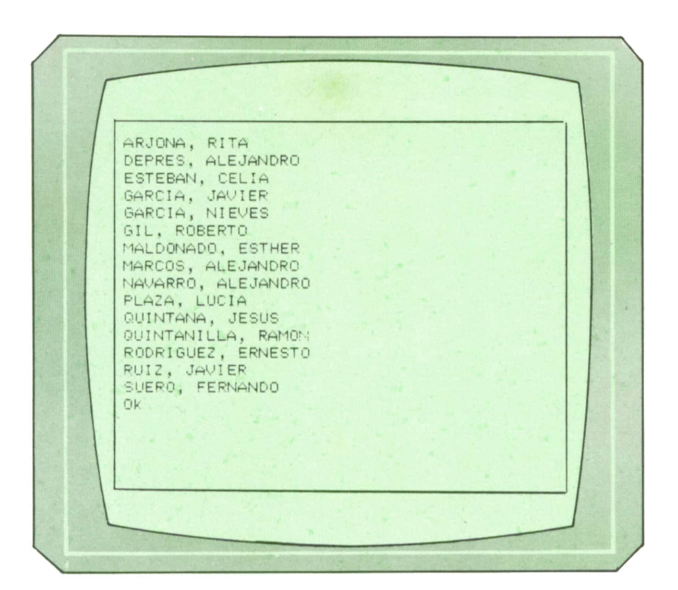
El método consiste en hacer sucesivas rondas de comparaciones contabilizadas por el bucle de las líneas 130-190 (con índice I). En cada ronda se hacen N-i (bucle con índice J) comparaciones de cada elemento de la lista con el inmediatamente posterior, procediendo al intercambio en caso de que el primer elemento de la comparación sea mayor (alfabéticamente hablando) que el segundo. De este modo la lista se ordena empezando por el último elemento y terminando por el primero.

En la figura 3 podemos ver de forma esquemática el proceso de ordenación para una lista numérica de cinco ele-






Representación esquemática del proceso de ordenación de una lista.




Presentación en pantalla del programa 3.

mentos. Al finalizar cada ronda queda ordenado el elemento señalado.

En la figura 4 podemos ver el aspecto de la pantalla tras una posible ejecución del programa 3.

Evidentemente, si en lugar de una matriz alfanumérica hubiésemos utilizado una numérica, el programa 3 serviría para ordenar de forma creciente una lista de números.

Hay que advertir que si tratamos de usar este programa en un SPECTRUM, comprobaremos con sorpresa que sólo se almacena en memoria la inicial de cada nombre. Esto es debido a que el SPECTRUM dimensiona las matrices alfanuméricas de una forma particular, con el siguiente formato:

DIM <nombre de variable> (n.º de elementos, longitud máxima), de modo que la línea 80 del programa 3 sería:

```
80 DIM L$ (N,20)
```

donde se establece una longitud máxima de 20 caracteres para cada nombre.

Para finalizar vamos a ver un ejemplo en el que se utilizan dos matrices alfanuméricas. Además ambas matrices se cargan mediante la lectura de datos almacenados en líneas DATA. El programa 4 tiene por objeto practicar geografía, permitiendo seleccionar el continente que deseemos. El ordenador irá presentando

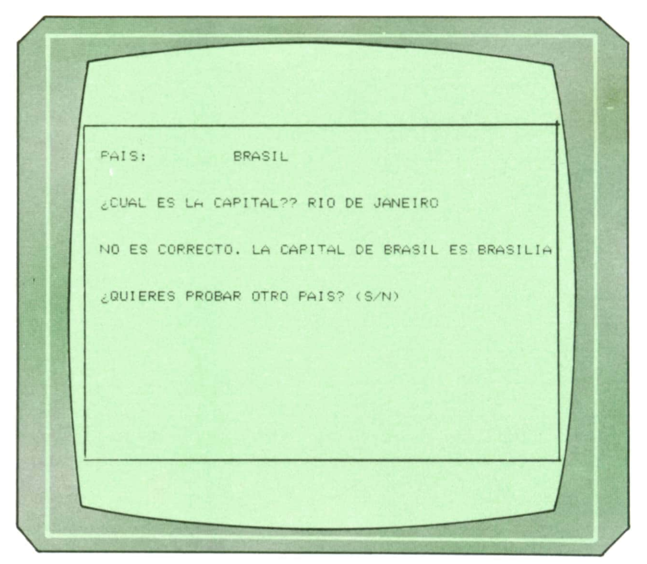
```
10 REM ************************
20 REM *  PAISES Y CAPITALES   *
30 REM ************************
40 CLS
50 DIM P$(5),C$(5)
60 PRINT TAB(16);"OPCIONES"
70 PRINT TAB(16);"--------"
80 PRINT :PRINT
90 PRINT TAB(16);"1. EUROPA":PRINT
100 PRINT TAB(16);"2. AMERICA":PRINT
110 PRINT TAB(16);"3. ASIA":PRINT
120 PRINT :PRINT
130 PRINT "PULSA LA OPCION DESEADA"
140 LET A$=INKEY$:IF A$="" THEN GOTO 140
150 IF ASC(A$)<49 OR ASC(A$)>51 THEN GOTO 140
160 IF A$="1" THEN RESTORE 380
170 IF A$="2" THEN RESTORE 390
180 IF A$="3" THEN RESTORE 400
190 CLS
200 FOR I=1 TO 5
210 READ P$(I),C$(I)
220 NEXT I
230 LET N=INT(RND(1)*5)+1
240 PRINT "PAIS:",P$(N)
250 PRINT :PRINT
260 INPUT "¿CUAL ES LA CAPITAL?";R$
270 PRINT :PRINT
280 IF R$=C$(N) THEN PRINT "CORRECTO":GOTO 300
290 PRINT "NO ES CORRECTO. LA CAPITAL DE ";P$(N);" ES ";C$(N)
300 PRINT :PRINT
310 PRINT "¿QUIERES PROBAR OTRO PAIS? (S/N)"
320 LET A$=INKEY$:IF A$="" THEN GOTO 320
330 IF A$="S" OR A$="s" THEN CLS:GOTO 230
340 CLS
350 PRINT "¿QUIERES CAMBIAR DE CONTINENTE? (S/N)"
360 LET A$=INKEY$:IF A$="" THEN GOTO 360
370 IF A$="S" OR A$="s" THEN CLS:GOTO 60
380 DATA ESPAÑA,MADRID,FRANCIA,PARIS,ITALIA,ROMA,AUSTRIA,VIENA,SUIZA,BERNA
390 DATA ESTADOS UNIDOS,WASHINGTON,BRASIL,BRASILIA,PERU,LIMA,ARGENTINA,BUENOS AI
RES,CUBA,LA HABANA
400 DATA JAPON,TOKIO,CHINA,PEKIN,INDIA,NUEVA DELHI,COREA,SEUL,THAILANDIA,BANGKOK
```

en pantalla sucesivos países del continente seleccionado a la vez que pregunta su capital.

En el programa sólo hemos puesto tres continentes y cinco países por continente para que el listado no resultara demasiado largo. Sin embargo, con un poco de paciencia no resulta muy difícil transformarlo en un auténtico programa educativo. Sólo habrá que dimensionar matrices más grandes, añadir los continentes que faltan y aumentar el número de líneas DATA.

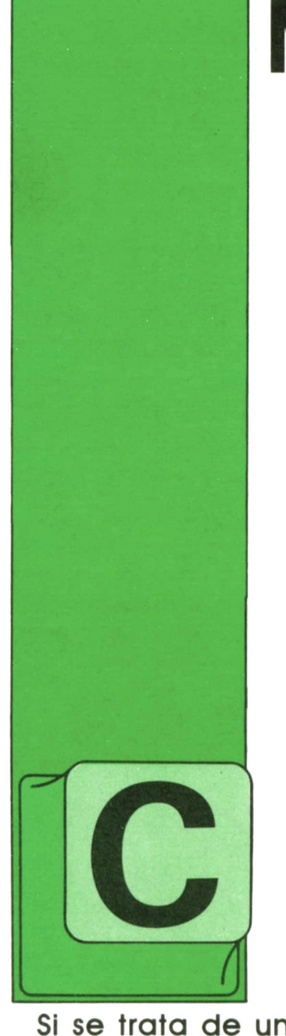
Por otra parte, este mismo programa podría servir para practicar muchas otras materias con unas modificaciones mínimas y cambiando las líneas DATA. Por ejemplo, podríamos practicar inglés poniendo en las líneas DATA palabras en español con su correspondiente significado en inglés.

Por último, en la figura 5 podemos ver el aspecto de la pantalla durante la ejecución del programa 4.




Presentación en pantalla del programa 4.

MAQUINA 6502



Comandos de desplazamiento

CON este tipo de comandos se desplaza la configuración de bits de un registro en una posición, bien hacia la derecha, bien hacia la izquierda.

Si se trata de una posición de memoria, tendremos un comando del tipo (Read-Modify-Write), es decir, que primero el procesador lee la posición de memoria direccionada, después la modifica desplazando sus bits y, por último, devuelve un nuevo valor a dicha posición.

ASL. Produce un desplazamiento a la izquierda del byte direccionado. En la posición cero, que se libera por este comando, se inserta un "0" y el séptimo bit se transmite al carry flag.

Así, con el comando ASL A teniendo el ACU el valor %01100101 se obtendría la siguiente configuración:

%11001010 C=0

Vemos que el Carry flag está desactivado, puesto que el séptimo bit del ACU estaba a "0".

Una observación importante: si nos fijamos en el número que contenía el ACU antes de la operación (101) y lo comparamos con el valor después del comando ASL (202) vemos que se ha duplicado. Esto es algo lógico, puesto que los bits, como sabemos, van en potencias de 2, pero interesante cuando queramos multiplicar un número por dos.

LSR. Corresponde al comando anterior, pero el desplazamiento de los bits es ahora hacia la derecha. En el bit que ocupa la posición siete se inserta un "0" y el bit cero se transmite al Carry flag.

Así, con el comando LSR A, teniendo el ACU la configuración %10010110 (150), se obtendría la siguiente configuración:

%01001011 (75) C=0

Como el lector habrá notado, con el comando LSR se puede dividir un número por 2. Pero esto no es todo, se puede saber si dicho número es par o es impar consultando el valor del Carry flag después de ejecutarse el comando.

Si el número es par, el bit cero contendrá un "0", y al efectuar el comando LSR, ese "0" pasará al Carry flag. Si el número es impar, el bit cero contendrá un "1", que se transmitirá al Carry flag. Ahora basta hacer una comparación del tipo BCC o BCS y bifurcar después según nos convenga.

Si con el comando ASL o LSR direccionamos una posición de memoria, el contenido del ACU no se altera.

ROL. Con este comando se rotan cíclicamente los bits del ACU o de la posición de memoria direccionada hacia la izquierda. Además, se traslada el Carry flag a la posición cero del registro, y el bit siete del registro pasa al Carry flag. En

realidad es como si trabajáramos con 9 bits en vez de con 8.

Por ejemplo, si el ACU contiene el número %01001001, y además C=1, el comando ROL A conducirá a la siguiente configuración:

%10010011 C=0

ROR. Es el comando inverso al anterior, y desplaza cíclicamente hacia la derecha el contenido de un registro. El contenido del Carry flag se traslada al bit 7, y el contenido del bit cero pasa al Carry flag.

En todos los casos en los que se utilicen comandos de desplazamiento, se podrán activar los flags N, Z e Interrupt dependiendo del resultado de la operación.

Veamos, por último, un resumen de estos comandos con sus códigos y modos de direccionamiento.

Modo de direccionamiento	ASL	LSR	ROL	ROR
ACU	$OA	$4A	$2A	$6A
Absoluto	$OE	$4E	$2E	$6E
Zeropage	$O6	$46	$26	$66
Absoluto indexado por X	$1E	$5E	$3E	$7E
Absoluto indexado por Y	$16	$56	$36	$76



Comandos de subrutina

Es una técnica muy utilizada en BASIC, cuyas instrucciones son GOSUB y RETURN, y no menos importante en lenguaje máquina. Cuando se llama a una subrutina el procesador debe memorizar la posición en la que encuentra en ese momento, así como el valor de algunas variables que luego necesitará cuando retorne de la subrutina.

Para todo esto se reserva un campo especial dentro de la memoria del COMMODORE 64, llamado Stack (pila) o memoria tampón. Este área de memoria está comprendida entre $0100 (256) y $01ff(511). Para posicionarse dentro de este stack y leer o almacenar datos en él, existe un apuntador o stack pointer, que es uno de los registros del procesador y al cual nos referimos cuando describimos los registros internos del microprocesador.

El stack trabaja de tal manera que el último valor introducido en él será el primero en extraerse, con una «pila» de cosas unas encima de otras. Gracias a este principio es posible el anidamiento de subrutinas.

Los comandos para el manejo de subrutinas son JSR ($20) y RTS ($60).

Este último ya lo habíamos visto, puesto que los programas que hemos tratado son llamados desde el BASIC, y cuando finalizan deben llevar el comando RTS, para volver a él.



Comandos de stack

Se utilizan para intercambiar datos entre el stack y el ACU, o el stack y el registro de estado. Cuando los datos van al stack, el stack pointer se decrementa en una unidad. Cuando los datos salen del stack, el stack pointer se incrementa en una unidad.

PHA. Guarda el contenido del ACU en el stack. El ACU no se altera.

PHP. Guarda el contenido del registro de estado en el stack. No se altera el contenido de registro de estado.

PLA. Es el inverso al comando PHA. Se extrae un byte del stack en la posición a la que apunte el stack pointer y se carga al ACU. Dependiendo del valor cargado, podrán alterarse los flags N y Z.

PLP. Se extrae un byte del stack y se eleva al registro de estado. Es el comando inverso al PHP.

Los códigos correspondientes a estos comandos son los siguientes:

Comando	Código
PHA	$48
PHP	$08
PLA	$68
PLP	$28



Comandos para las interrupciones

Son los últimos comandos que vamos a ver en esta sección dedicada al código máquina.

La CPU del COMMODORE 64 tiene dos líneas de interrupciones (sin contar la de Reset), que se suelen denominar IRQ y NMI.

Además existe la interrupción por programa BRK.

1. **Rutina de Reset.** Consiste en una serie de inicializaciones que sirven para poner en funcionamiento al unísono todos y cada uno de los circuitos integrantes de su C-64.

Al conectar el ordenador, el procesador se dirige a las posiciones $FFFC (65562) y $FFFD (65563) en las que se encuentra un apuntador conteniendo los números $E2 y $FC, respectivamente. Es decir, apunta a la dirección $FCE2 (64738), donde realmente empieza la rutina de Reset.

Como el apuntador y la propia rutina están en ROM (Read Only Memory), no pueden en principio ser alterados, a no ser que copiemos la ROM del sistema operativo a la RAM que hay debajo de ella, y a continuación desconectemos dicha ROM mediante la posición en memoria 1, poniendo el valor 53 en decimal.

Ahora esta rutina se ejecuta en RAM y aquí sí puede ser modificada, o cambiando el vector a otra zona y crear otra rutina de Reset nueva.

2. **Interrupciones IRQ.** Son las que se explicaron anteriormente, ocurriendo cada sesentaavo de segundo. Son del tipo estable, es decir, pueden ser deshabilitadas y van a ser las más utilizadas en nuestros programas.

3. **Interrupciones NMI.** Por estas líneas se puede recibir una señal de interrupción de cualquier dispositivo que precise su servicio (reloj, bus, serie, teclado...), y produce una interrupción de lo que se encuentre ejecutando para atender la llamada. Este tipo de interrupciones son del tipo no evitable, es decir, se producen siempre que haya un dispositivo digamos «pidiendo la interrupción».

4. **Interrupciones BRK.** Son interrupciones solicitadas dentro de nuestro programa en lenguaje máquina mediante la instrucción correspondiente: BRK($00) y RTI($40).

Al ser llamada cualquiera de las interrupciones anteriores, al procesador se dirige a donde señale el apuntador correspondiente.

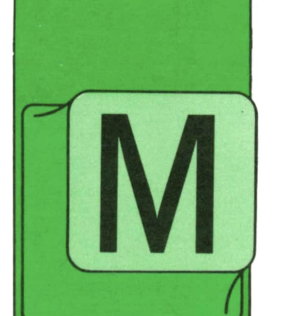
Interrupción	Apuntador
Reset	$FFFC-$FFFD
IRQ	$0314-$0315
NMI	$0316- 0317
BRK	$0318- 0319

Para terminar mencionaremos un comando que no hace nada. No le extrañe, pues tiene una misión: quitar instrucciones de un programa sin tener que modificar éste, o para rellenar bucles de espera. Su símbolo es NOP y su código $EA.

PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS

Programa: Gráficas para AMSTRAD

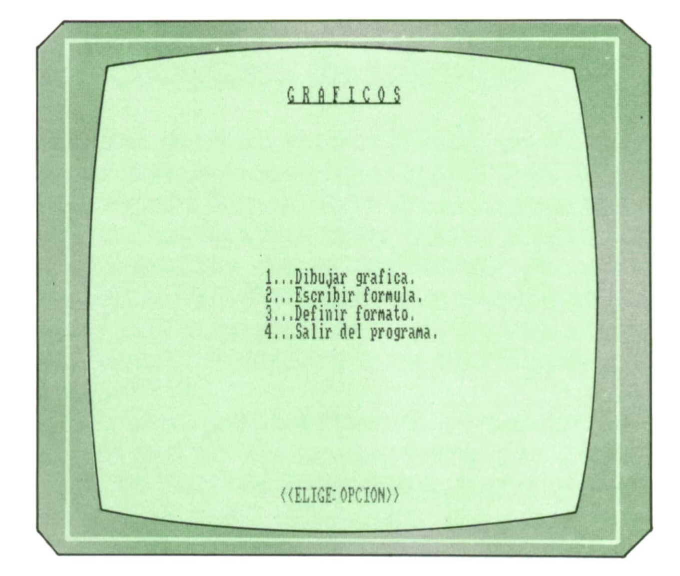


UCHAS veces, sobre todo los estudiantes de ciencias, necesitan tener a mano un programa que les dibuje cualquier tipo de gráfica en la pantalla. Un programa que permita definir los límites de la gráfica, los incrementos, que nos dibuje las asíntotas, etc.

Con este programa podremos ver cualquier gráfica que nosotros queramos, por complicada que ésta sea, en pocos instantes.

Al principio nos aparece un menú con cuatro opciones:

1. *Dibujar gráfica.* Una vez que hemos definido cómo queremos la gráfica y qué función vamos a utilizar, debemos pulsar esta opción.
2. *Definir fórmula.* Introducimos la función a visualizar.
3. *Definir formato.* El ordenador nos hace una serie de preguntas sobre la gráfica, sobre los límites, etc.
4. *Salir del programa.*




Definición del formato.

FORMATO:

1...Punto o Trazo.(T)
2...Dibuja eje x. (S)
3...Dibuja eje y. (S)
4...STEP Auto. (S)
5...Escribe valores de x e y. (S)
6...Fn. Tri. Radianes o Grados.(G)
7...Margen cuando XY → ∞. (N)
8...Dibuja cuadricula. (N)


Menú principal.

El ordenador nos dibujará la gráfica en la parte derecha de la pantalla mientras nos informa de lo que está haciendo en la parte izquierda.

```
10 REM ******************************
20 REM ***       GRAFICAS!        ***
30 REM *** Un programa realizado ***
40 REM ***          Por           ***
50 REM *** Carlos A. Maria Morin ***
60 REM ***                        ***
70 REM ***     (C) Ediciones      ***
80 REM ***  Siglo Cultural-1987   ***
90 REM ******************************
100 REM
110 REM ********************************
120 REM ******** INICIALIZACION ********
130 REM ********************************
140 REM
150 BORDER 1
160 PAPER 0
170 PEN 5
180 CLS
190 MODE 0
200 LOCATE 3,12
210 PRINT"G R A F I C O S"
220 FOR ret=1 TO 3000
230 NEXT
240 MODE 2
250 DIM aumento$(3)
260 A1=1
270 A2=1
280 X5=0
290 aumento=1
300 k1$(1)="T"
310 k1$(2)="S"
320 K1$(3)="S"
330 k1$(4)="S"
340 k1$(5)="S"
350 k1$(6)="G"
360 k1$(7)="N"
370 k1$(8)="N"
380 SYMBOL 244,0,0,60,66,129,129,66,60
390 SYMBOL 245,0,0,60,66,129,129,66,60
400 WINDOW #1,1,40,25,25
410 KEY 156,CHR$(13)+"goto 380"+CHR$(13)
420 KEY 157,"cls:list"+CHR$(13)
430 KEY DEF 15,0,48,156,157
440 REM
450 REM ******************************
460 REM ** PROGRAMA PRINCIPAL 'MENU' **
470 REM ******************************
480 REM
490 CLS
500 BORDER 24
510 CLS
520 PRINT CHR$(22);CHR$(1)
530 LOCATE 32,2
540 PRINT" G R A F I C O S "
550 LOCATE 32,2
560 PRINT" _______________ "
570 PRINT CHR$(22);CHR$(0)
580 PRINT:PRINT:PRINT
590 DEF FNy(x)=((x*(SQR(1-x)))/(SQR(x+1)))
600 PRINT:PRINT:PRINT:PRINT:PRINT
610 PRINT SPACE$(29);"1...Dibujar grafica."
620 PRINT SPACE$(29);"2...Escribir formula."
630 PRINT SPACE$(29);"3...Definir formato."
640 PRINT SPACE$(29);"4...Salir del programa."
650 PRINT:PRINT:PRINT
660 PRINT:PRINT:PRINT
670 LOCATE 31,24
680 PRINT " <<ELIGE OPCION>>  "
```

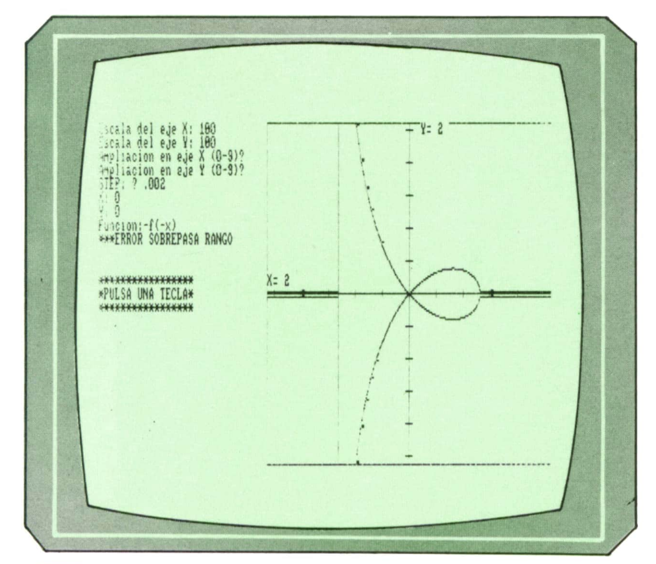
```
690 k$=INKEY$
700 IF k$="" THEN 690 ELSE k=ASC(k$)-48
710 IF k<1 OR k>4 THEN SOUND 1,60,10,15 ELSE 780
720 LOCATE 31,24
730 PRINT "<<OPCION INVALIDA>>"
740 CLEAR INPUT
750 FOR ret=1 TO 1500
760 NEXT
770 GOTO 670
780 ON k GOSUB 1120,1040,840,800
790 GOTO 490
800 CLS
810 BORDER 1
820 END
830 REM
840 REM ******************************
850 REM **** SELECCION DEL FORMATO ****
860 REM ******************************
870 REM
880 CLS
890 PRINT:PRINT"FORMATO:"
900 PRINT:PRINT:RESTORE
910 FOR f=1 TO 8
920 READ af$,neg$,men$
930 PRINT men$;
940 IF f=7 THEN PRINT "Margen cuando XY "CHR$(154)+CHR$(243)" "CHR$(244)+CHR$(24
5)".     (";
950 PRINT k1$(f)") ";
960 k$=INKEY$
970 IF k$="" THEN 960 ELSE k$=UPPER$(k$)
980 IF k$<>"" THEN IF k$=af$ OR k$=neg$ THEN k1$(f)=k$
990 IF k$<>af$ AND k$<>neg$ THEN PRINT ELSE PRINT k1$(f)
1000 NEXT f
1010 RETURN
1020 STOP
1030 REM
1040 REM ======== VOLVER AL MENU ========
1050 REM
1060 CLS
1070 LOCATE 21,24
1080 PRINT"Cambiar la ecuacion y pulsar  [SHIFT] y [0]"
1090 LOCATE 20,10
1100 EDIT 590
1110 GOTO 490
1120 CLS
1130 PAPER 2
1140 PEN 5
1150 BORDER 1
1160 CLS
1170 IF k1$(6)="R" THEN RAD ELSE DEG
1180 REM
1190 REM ********************************
1200 REM ********** DIBUJA EJES **********
1210 REM ********************************
1220 REM
1230 ORIGIN 440,200
1240 MOVE -200,0:DRAW -200,200
1250 MOVE -200,199:DRAW 200,199
1260 MOVE 199,199:DRAW 200,-200
1270 MOVE 0,0:MOVE -200,0
1280 DRAW -200,-199:MOVE -200,-200
1290 DRAW 200,-200:MOVE 199,-200
1300 DRAW 199,200:MOVE 5,38
1310 DRAW -5,38:MOVE 5,76
1320 DRAW -5,76:MOVE 5,114
1330 DRAW -5,114
1340 IF k1$(2)="S" THEN MOVE 200,0:DRAW -200,0
1350 MOVE 5,152:DRAW -5,152
```

```
1360 MOVE 5,190:DRAW -5,190
1370 MOVE 5,-38 :DRAW -5,-38
1380 MOVE 5,-76:DRAW -5,-76
1390 MOVE 5,-114:DRAW -5,-114
1400 MOVE 5,-152:DRAW -5,-152
1410 MOVE 5,-190:DRAW -5,-190
1420 PLOT 0,0
1430 IF k1$(3)="S" THEN MOVE 0,200:DRAW 0,-200
1440 MOVE 38,5:DRAW 38,-5
1450 MOVE 76,5:DRAW 76,-5
1460 MOVE 114,5:DRAW 114,-5
1470 MOVE 152,5:DRAW 152,-5
1480 MOVE 190,5:DRAW 190,-5
1490 MOVE -38,5:DRAW -38,-5
1500 MOVE -76,5:DRAW -76,-5
1510 MOVE -114,5:DRAW -114,-5
1520 MOVE -152,5:DRAW -152,-5
1530 MOVE -190,5:DRAW -190,-5
1540 IF k1$(8)="S" THEN GOSUB 3430
1550 REM
1560 REM ********************************
1570 REM * SELECCION DE ESCALA Y AUMENTO *
1580 REM ********************************
1590 REM
1600 LOCATE 1,1
1610 PRINT "Escala del eje X:";A1;
1620 INPUT ;A1$
1630 IF A1$<>"" THEN A1=VAL(A1$)
1640 LOCATE 18,1
1650 PRINT A1;"      "
1660 LOCATE 1,2
1670 PRINT"Escala del eje Y:";A2;
1680 INPUT A2$
1690 IF A2$<>"" THEN A2=VAL(A2$)
1700 LOCATE 18,2
1710 PRINT A2;"      "
1720 LOCATE 31,12
1730 PRINT"X=";INT(1000*200/A1)/1000
1740 LOCATE 58,1
1750 PRINT"Y=";INT(1000*200/A2)/1000
1760 LOCATE 1,3
1770 PRINT"Ampliacion en eje X (0-9)";
1780 INPUT aumento$(1)
1790 LOCATE 27,3
1800 PRINT " ";aumento$(1)
1810 IF aumento$(1)<>"" AND aumento$(1)<>"0" THEN 1820 ELSE 1870
1820 IF aumento$(1)<>"1" AND aumento$(1)<>"2" THEN 1830 ELSE 1870
1830 IF aumento$(1)<>"3" AND aumento$(1)<>"4" THEN 1840 ELSE 1870
1840 IF aumento$(1)<>"5" AND aumento$(1)<>"6" THEN 1850 ELSE 1870
1850 IF aumento$(1)<>"7" AND aumento$(1)<>"8" THEN 1860 ELSE 1870
1860 IF aumento$(1)<>"9" THEN 1760 ELSE 1870
1870 LOCATE 1,4
1880 PRINT"Ampliacion en eje Y (0-9)";
1890 INPUT aumento$(2)
1900 LOCATE 27,4
1910 PRINT " ";aumento$(2)
1920 IF aumento$(2)<>"" AND aumento$(2)<>"0" THEN 1930 ELSE 1980
1930 IF aumento$(2)<>"1" AND aumento$(2)<>"2" THEN 1940 ELSE 1980
1940 IF aumento$(2)<>"3" AND aumento$(2)<>"4" THEN 1950 ELSE 1980
1950 IF aumento$(2)<>"5" AND aumento$(2)<>"6" THEN 1960 ELSE 1980
1960 IF aumento$(2)<>"7" AND aumento$(2)<>"8" THEN 1970 ELSE 1980
1970 IF aumento$(2)<>"9" THEN 1870 ELSE 1980
1980 IF aumento$(1)="" THEN aumento(1)=1
1990 IF aumento$(1)="0" THEN aumento(1)=10
2000 IF aumento$(1)="1" THEN aumento(1)=20
2010 IF aumento$(1)="2" THEN aumento(1)=30
2020 IF aumento$(1)="3" THEN aumento(1)=40
2030 IF aumento$(1)="4" THEN aumento(1)=50
2040 IF aumento$(1)="5" THEN aumento(1)=60
```

```
2050 IF aumento$(1)="6" THEN aumento(1)=70
2060 IF aumento$(1)="7" THEN aumento(1)=80
2070 IF aumento$(1)="8" THEN aumento(1)=90
2080 IF aumento$(1)="9" THEN aumento(1)=100
2090 IF aumento$(2)="" THEN aumento(2)=1
2100 IF aumento$(2)="0" THEN aumento(2)=10
2110 IF aumento$(2)="1" THEN aumento(2)=20
2120 IF aumento$(2)="2" THEN aumento(2)=30
2130 IF aumento$(2)="3" THEN aumento(2)=40
2140 IF aumento$(2)="4" THEN aumento(2)=50
2150 IF aumento$(2)="5" THEN aumento(2)=60
2160 IF aumento$(2)="6" THEN aumento(2)=70
2170 IF aumento$(2)="7" THEN aumento(2)=80
2180 IF aumento$(2)="8" THEN aumento(2)=90
2190 IF aumento$(2)="9" THEN aumento(2)=100
2200 IF k1$(4)="N" THEN LOCATE 1,5:INPUT "STEP: ";S
2210 IF k1$(4)="N" THEN ELSE S=2/A1:LOCATE 1,5:PRINT"STEP:";"Autom.";S;"  "
2220 IF S=0 THEN S=2/A1
2230 IF k1$(1)="T" THEN 2820 ELSE 2260
2240 END
2250 REM
2260 REM ************************************
2270 REM * TRATAMIENTO GRAFICO DE FUNCIONES *
2280 REM ************************************
2290 REM
2300 ON ERROR GOTO 3380
2310 LOCATE 1,8
2320 PRINT"Funcion:f(x)"
2330 MOVE X5,0
2340 FOR X=X5 TO 200/A1 STEP S
2350 IF k1$(5)="S" THEN LOCATE 1,6:PRINT "X:";XPOS;:LOCATE 1,7:PRINT"Y:";YPOS"
    "
2360 IF X*A1>32750 OR Y*A2>32750 OR X*A1<-32750 OR Y*A2<-32750 THEN 2410
2370 IF k1$(7)="N" THEN IF XPOS>205 OR XPOS<-205 OR YPOS>205 OR YPOS<-205 THEN 2
420
2380 px=X*A1*aumento(1)
2390 py=INT(FN y(X)*A2*aumento(2))
2400 PLOT px,py
2410 NEXT
2420 LOCATE 1,8
2430 PRINT"Funcion:f(-x)"
2440 MOVE -200/A1,0
2450 FOR X=(-200/A1)/aumento(2) TO X5 STEP S
2460 IF k1$(5)="S" THEN LOCATE 1,6:PRINT "X:";XPOS;:LOCATE 1,7:PRINT"Y:";YPOS"
    "
2470 IF X*A1>32750 OR Y*A2>32750 OR X*A1<-32750 OR Y*A2<-32750 THEN 2520
2480 IF k1$(7)="N" THEN IF XPOS>205 OR XPOS<-205 OR YPOS>205 OR YPOS<-205 THEN 2
530
2490 px=X*A1*aumento(1)
2500 py=INT(FN y(X)*A2*aumento(2))
2510 PLOT px,py
2520 NEXT
2530 LOCATE 1,8
2540 PRINT"Funcion:-f(x)"
2550 MOVE X5,0
2560 FOR X=X5 TO 200/A1 STEP S
2570 IF k1$(5)="S" THEN LOCATE 1,6:PRINT "X:";XPOS;:LOCATE 1,7:PRINT"Y:";YPOS"
    "
2580 IF X*A1>32750 OR Y*A2>32750 OR X*A1<-32750 OR Y*A2<-32750 THEN 2630
2590 IF k1$(7)="N" THEN IF XPOS>205 OR XPOS<-205 OR YPOS>205 OR YPOS<-205 THEN 2
640
2600 px=X*A1*aumento(1)
2610 py=INT(-(FN y(X)*A2*aumento(2)))
2620 PLOT px,py
2630 NEXT
2640 LOCATE 1,8
2650 PRINT"Funcion:-f(-x)"
2660 MOVE -200/A1,0
2670 FOR X=((-200/A1)/aumento(2)) TO X5 STEP S
```

```
2680 IF k1$(5)="S" THEN LOCATE 1,6:PRINT "X:";XPOS;:LOCATE 1,7:PRINT"Y:";YPOS"
     "
2690 IF X*A1>32750 OR Y*A2>32750 OR X*A1<-32750 OR Y*A2<-32750 THEN 2740
2700 IF k1$(7)="N" THEN IF XPOS>205 OR XPOS<-205 OR YPOS>205 OR YPOS<-205 THEN 2
750
2710 px=X*A1*aumento(1)
2720 py=INT(-(FN y(X)*A2*aumento(2)))
2730 PLOT px,py
2740 NEXT
2750 LOCATE 1,12
2760 PRINT"*****************"
2770 LOCATE 1,13
2780 PRINT"*PULSA UNA TECLA*"
2790 LOCATE 1,14:PRINT"*****************"
2800 CALL &BB06:REM EN ESPERA DE QUE SE PULSE UNA TECLA
2810 GOTO 490
2820 ON ERROR GOTO 3380
2830 LOCATE 1,8
2840 PRINT"Funcion:f(x)"
2850 MOVE X5,0
2860 FOR X=X5 TO 200/A1 STEP S
2870 IF k1$(5)="S" THEN LOCATE 1,6:PRINT "X:";XPOS;:LOCATE 1,7:PRINT"Y:";YPOS"
     "
2880 IF X*A1>32750 OR Y*A2>32750 OR X*A1<-32750 OR Y*A2<-32750 THEN 2930
2890 IF k1$(7)="N" THEN IF XPOS>205 OR XPOS<-205 OR YPOS>205 OR YPOS<-205 THEN
2940
2900 px=X*A1*aumento(1)
2910 py=INT(FN y(X)*A2*aumento(2))
2920 DRAW px,py
2930 NEXT
2940 LOCATE 1,8
2950 PRINT"Funcion:f(-x)"
2960 MOVE -200,0
2970 FOR X=(-200/A1)/aumento(2) TO X5 STEP S
2980 IF k1$(5)="S" THEN LOCATE 1,6:PRINT "X:";XPOS;:LOCATE 1,7:PRINT"Y:";YPOS"
     "
2990 IF X*A1>32750 OR Y*A2>32750 OR X*A1<-32750 OR Y*A2<-32750 THEN 3040
3000 IF k1$(7)="N" THEN IF XPOS>205 OR XPOS<-205 OR YPOS>205 OR YPOS<-205 THEN 3
050
3010 px=X*A1*aumento(1)
3020 py=INT(FN y(X)*A2*aumento(2))
3030 DRAW px,py
3040 NEXT
3050 LOCATE 1,8
3060 PRINT"Funcion:-f(x)"
3070 MOVE X5,0
3080 FOR X=X5 TO 200/A1 STEP S
3090 IF k1$(5)="S" THEN LOCATE 1,6:PRINT "X:";XPOS;:LOCATE 1,7:PRINT"Y:";YPOS"
     "
3100 IF X*A1>32750 OR Y*A2>32750 OR X*A1<-32750 OR Y*A2<-32750 THEN 3150
3110 IF k1$(7)="N" THEN IF XPOS>205 OR XPOS<-205 OR YPOS>205 OR YPOS<-205 THEN
160
3120 px=X*A1*aumento(1)
3130 py=INT(-(FN y(X)*A2*aumento(2)))
3140 DRAW px,py
3150 NEXT
3160 LOCATE 1,8
3170 PRINT"Funcion:-f(-x)"
3180 MOVE -200,0
3190 FOR X=(-200/A1)/aumento(2) TO X5 STEP S
3200 IF k1$(5)="S" THEN LOCATE 1,6:PRINT "X:";XPOS;:LOCATE 1,7:PRINT"Y:";YPOS"
     "
3210 IF X*A1>32750 OR Y*A2>32750 OR X*A1<-32750 OR Y*A2<-32750 THEN 3260
3220 IF k1$(7)="N" THEN IF X>205 OR X<-205 OR YPOS>205 OR YPOS<-205 THEN 3270
3230 px=X*A1*aumento(1)
3240 py=INT(-(FN y(X)*A2*aumento(2)))
3250 DRAW px,py
3260 NEXT
3270 LOCATE 1,12
```

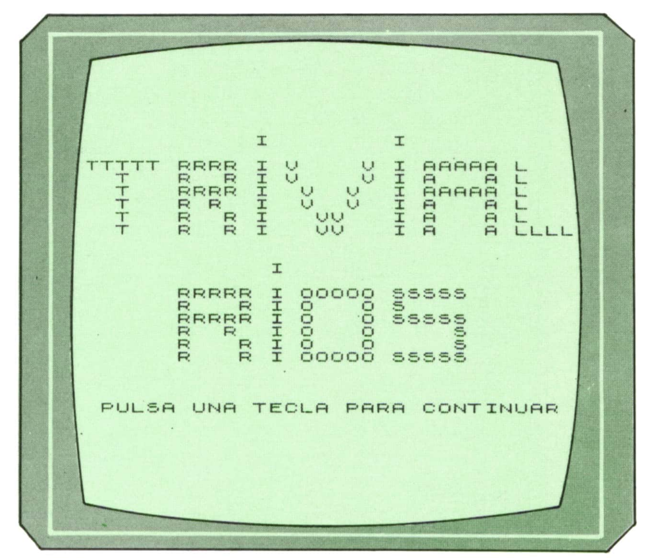
```
3280 PRINT"****************"
3290 LOCATE 1,13
3300 PRINT"*PULSA UNA TECLA*"
3310 LOCATE 1,14
3320 PRINT"****************"
3330 CALL &BB06:REM EN ESPERA DE QUE SE PULSE UNA TECLA
3340 GOTO 490
3350 REM
3360 REM ===== TRATAMIENTO DE ERRORES =====
3370 REM
3380 LOCATE 1,9
3390 PRINT "***ERROR SOBREPASA RANGO"
3400 RESUME NEXT
3410 END
3420 REM
3430 REM *********************************
3440 REM ******* DIBUJA CUADRICULA *******
3450 REM *********************************
3460 REM
3470 ORIGIN 440,200
3480 MOVE 200,38:DRAW -200,38
3490 MOVE 200,76:DRAW -200,76
3500 MOVE 200,114:DRAW -200,114
3510 MOVE 200,152:DRAW -200,152
3520 MOVE 200,190:DRAW -200,190
3530 MOVE 200,-38 :DRAW -200,-38
3540 MOVE 200,-76:DRAW -200,-76
3550 MOVE 200,-114:DRAW -200,-114
3560 MOVE 200,-152:DRAW -200,-152
3570 MOVE 200,-190:DRAW -200,-190
3580 MOVE 38,200:DRAW 38,-200
3590 MOVE 76,200:DRAW 76,-200
3600 MOVE 114,200:DRAW 114,-200
3610 MOVE 152,200:DRAW 152,-200
3620 MOVE 190,200:DRAW 190,-200
3630 MOVE -38,200:DRAW -38,-200
3640 MOVE -76,200:DRAW -76,-200
3650 MOVE -114,200:DRAW -114,-200
3660 MOVE -152,200:DRAW -152,-200
3670 MOVE -190,200:DRAW -190,-200
3680 RETURN
3690 END
3700 REM
3710 REM *********************************
3720 REM ************ DATAS *************
3730 REM *********************************
3740 REM
3750 DATA P,T,1...Punto o Trazo.(
3760 DATA S,N,2...Dibuja eje x.  (
3770 DATA S,N,3...Dibuja eje y.  (
3780 DATA S,N,4...STEP   Auto.   (
3790 DATA S,N,5...Escribe valores de x e y.  (
3800 DATA R,G,6...Fn. Tri. Radianes o Grados.(
3810 DATA S,N,7...
3820 DATA S,N,8...Dibuja cuadricula.         (
```




Dibujo de la función.

y = f(−x)

$f(x) = ((x\sqrt{1 - x})/\sqrt{x + 1})$




Presentación del programa.

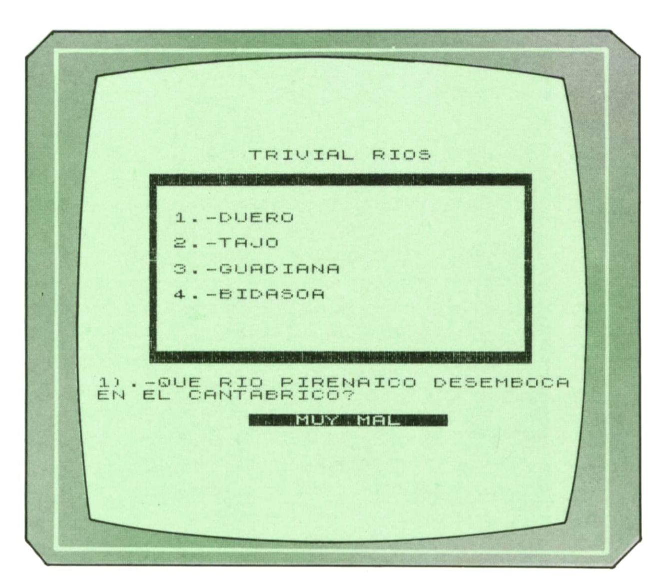


Programa: TRIVIAL RIOS para SPECTRUM

A continuación aparece la versión del programa TRIVIAL RIOS que ya apareció para otros ordenadores y que ahora aparece para el SPECTRUM.

El programa tiene 54 preguntas distintas que nos irán apareciendo una a una. Tras la pregunta nos aparecerán cuatro posibles respuestas y nosotros tendremos que pulsar la adecuada.

Una vez terminada la sesión de preguntas el ordenador nos dirá qué tal lo hemos hecho y cuál ha sido nuestra puntuación.




El programa preguntándonos.

```
 10 REM  ***********************
 20 REM  **** TRIVIAL RIOS ****
 30 REM  ***********************
 40 REM  ** POR J.M.GUTIERREZ *
 50 REM  ***********************
 51 REM
 52 REM *********************************
 53 REM * (c) Ediciones Siglo Cultural *
 54 REM * (c) 1987                      *
 55 REM *********************************
 56 REM
 60 POKE USR "N"+0,BIN 01111110
 70 POKE USR "N"+1,BIN 0
 80 DIM t(75)
 90 DIM a(3)
100 DIM r$(12,20)
110 CLS
```

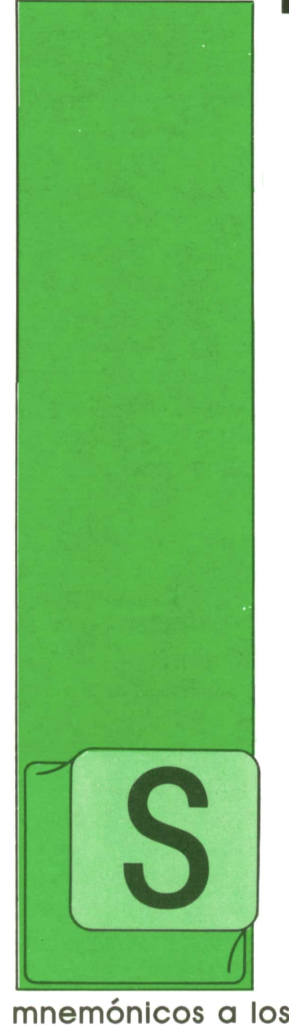
```
120 PRINT "           I        I"
130 PRINT
140 PRINT "TTTTT RRRR I V    V I AAAAA L    "
150 PRINT "  T   R  R I V    V I A   A L    "
160 PRINT "  T   RRRR I  V  V  I AAAAA L    "
170 PRINT "  T   R R  I  V  V  I A   A L    "
180 PRINT "  T   R  R I   VV   I A   A L    "
190 PRINT "  T   R  R I   VV   I A   A LLLL"
200 PRINT
210 PRINT
220 PRINT "            I"
230 PRINT
240 PRINT "      RRRRR I OOOOO SSSSS"
250 PRINT "      R   R I O   O S"
260 PRINT "      RRRRR I O   O SSSSS"
270 PRINT "      R  R  I O   O     S"
280 PRINT "      R   R I O   O     S"
290 PRINT "      R   R I OOOOO SSSSS"
291 REM
292 REM ******************************************************
300 REM * CONTADORES DE RESPUESTAS ACERTADAS Y EQUIVOCADAS *
301 REM ******************************************************
302 REM
310 FOR i=1 TO 25
320    BEEP .05,-30+INT (RND*50)
330 NEXT i
340 PRINT  FLASH 1;AT 21,1;"PULSA UNA TECLA PARA CONTINUAR"
350 LET k$=INKEY$
360 IF k$="" THEN  GO TO 350
370 CLS
380 PRINT  FLASH 1;AT 5,0;"CARGANDO DATOS,POR FAVOR ESPERE"
381 REM
382 REM ***********************************
390 REM * CARGA DE LOS DATOS DE LOS RIOS *
391 REM ***********************************
392 REM
400 GO SUB 1150
401 REM
402 REM **************************************************
410 REM * CARGA DE LOS DATOS DE LA TABLA DE RESPUESTAS *
411 REM **************************************************
412 REM
420 GO SUB 2040
421 REM
422 REM *********************************************************************
430 REM * EL SIGUIENTE BUCLE FOR SE EJECUTA TANTAS VECES COMO PREGUNTAS HAY *
431 REM *********************************************************************
432 REM
438 REM *********************************
440 REM * EN ESTE CASO HAY 54 PREGUNTAS *
441 REM *********************************
442 REM
450 LET bi=0
460 LET ma=0
470 RESTORE 1480
480 FOR i=1 TO 54
481 REM
482 REM *********************************************
490 REM * IR A LA SUBRUTINA DE DIBUJO DE LA VENTANA *
491 REM *********************************************
492 REM
500 CLS
510 PRINT AT 0,10; FLASH 1;"TRIVIAL RIOS"
520    GO SUB 2240
530    READ p$
540    LET re=t(i)
550    RANDOMIZE PEEK 23672
560    LET a(1)=INT (RND*12)+1
570    IF a(1)=re THEN  GO TO 550
```

```
 580     RANDOMIZE PEEK 23672
 590     LET a(2)=INT (RND*12)+1
 600     IF a(2)=a(1) OR a(2)=re THEN  GO TO 580
 610     RANDOMIZE PEEK 23672
 620     LET a(3)=INT (RND*12)+1
 630     IF a(3)=a(2) OR a(3)=a(1) OR a(3)=re THEN  GO TO 610
 640     LET po=INT (RND*4)+1
 645 REM
 650 REM *** PO ES LA POSICION EN LA VENTANA DE LA RESPUESTA ***
 655 REM
 660     PRINT AT 3+po*2,5;po;".-";r$(re)
 670     LET k=1
 680       FOR j=1 TO 3
 690          IF po=j THEN  GO TO 710
 700          GO TO 740
 710          LET k=k+1
 720          PRINT AT 3+k*2,5;k;".-";r$(a(j))
 730          GO TO 750
 740          PRINT AT 3+k*2,5;k;".-";r$(a(j))
 750          LET k=k+1
 760       NEXT j
 770     PRINT AT 18,0;i;").-";p$
 780     INPUT c$
 785 REM
 790 REM *** C$ ES LA CONTESTACION DADA A UNA PREGUNTA ***
 795 REM
 800     IF LEN c$>1 OR c$="" THEN  BEEP .30,20: GO TO 770
 810     IF c$<"0" OR c$>"9" THEN  BEEP .30,20: GO TO 770
 820     IF VAL c$<1 OR VAL c$>4 THEN  BEEP .30,20: GO TO 770
 830     IF VAL c$=po THEN  GO TO 890
 840     BEEP .20,-30
 850     PRINT  INVERSE 1;AT 21,10;"   MUY MAL    "
 860     LET ma=ma+1
 870     PRINT AT 3+po*2,5; FLASH 1;po; FLASH 0;".-";r$(re)
 880     GO TO 920
 890     BEEP .20,30
 900     PRINT AT 21,10; INVERSE 1;"   MUY BIEN    "
 910     LET bi=bi+1
 920     FOR f=1 TO 200
 930     NEXT f
 940     PRINT AT 21,7; INVERSE 1;"SPACE"; INVERSE 0;" PARA TERMINAR"
 950     LET k$=INKEY$
 960     IF k$="" THEN  GO TO 950
 970     IF k$=" " THEN  GO TO 1000
 980 NEXT i
 990 LET i=i-1
1000 CLS
1010 PRINT AT 0,5; INVERSE 1; FLASH 1;"RESULTADO DEL CONTROL"
1020 PRINT AT 3,0;"respuestas acertadas     --> ";bi
1030 PRINT AT 5,0;"respuestas falladas      --> ";ma
1040 PRINT AT 7,0;"preguntas hechas         --> ";i
1050 PRINT AT 9,0; BRIGHT 1;"porcentaje hidrografico --> ";INT ((bi/i)*100);"%"
1060 PRINT AT 12,0;"QUIERES EMPEZAR DE NUEVO(S/N) ?"
1070 LET k$=INKEY$
1080 IF k$="" THEN  GO TO 1070
1090 IF k$="S" OR k$="s" THEN  GO TO 430
1100 CLS
1110 GO TO 2380
1111 REM
1120 REM ************************************************
1130 REM * DATAS DE LOS PRINCIPALES RIOS DE LA PENINSULA *
1140 REM ************************************************
1141 REM
1150 DATA "MI"+CHR$ 157+"O"
1160 DATA "DUERO"
1170 DATA "TAJO"
1180 DATA "GUADIANA"
1190 DATA "GUADALQUIVIR"
1200 DATA "EBRO"
```

```
1210 DATA "JUCAR"
1220 DATA "SEGURA"
1230 DATA "NALON"
1240 DATA "NERVION"
1250 DATA "BIDASOA"
1260 DATA "TURIA"
1261 REM
1270 REM ************************
1280 REM * FIN DE DATAS DE RIOS *
1290 REM ************************
1300 REM
1320 REM ************************************
1330 REM * CREACION DE LA TABLA DE LOS RIOS *
1340 REM ************************************
1341 REM
1350 RESTORE 1150
1360 FOR i=1 TO 12
1370    READ r$(i)
1380 NEXT i
1390 RETURN
1391 REM
1400 REM **********************************
1410 REM * FIN DE LA CREACION DE LA TABLA *
1420 REM **********************************
1440 REM
1450 REM *************************
1460 REM * PREGUNTAS DE LOS RIOS *
1470 REM *************************
1471 REM
1480 DATA "QUE RIO PIRENAICO DESEMBOCA EN EL CANTABRICO?"
1490 DATA "QUE RIO DESEMBOCA EN PUENTE FORCINAS?"
1500 DATA "EL RIO CABRIEL ES UN AFLUEN-TE DEL..."
1510 DATA "EL RIO TAMBRE ES UN AFLUEN- TE DEL..."
1520 DATA "EL RIO GUADIANA MENOR ES UN AFLUENTE DEL..."
1530 DATA "QUE RIO DESEMBOCA EN OPORTO?"
1540 DATA "QUE RIO PASA POR BURGOS?"
1550 DATA "QUE RIO DESEMBOCA EN AYA -  MONTE (HUELVA) ?"
1560 DATA "QUE RIO DESEMBOCA EN LA    GUARDIA?"
1570 DATA "EL RIO GALLO ES UN AFLUENTE DEL..."
1580 DATA "EL RIO TORMES ES UN AFLUEN- TE DEL..."
1590 DATA "EL RIO ESLA ES UN AFLUENTE DEL..."
1600 DATA "QUE RIO PASA POR ORENSE?"
1610 DATA "QUE RIO PASA POR LA CIUDAD DE TOLEDO?"
1620 DATA "QUE RIO EN SU TRAMO FINAL  SEPARA ESPA"+CHR$ 157+"A DE FRANCIA?"
1630 DATA "QUE RIO NACE EN PE"+CHR$ 157+"A ORDU"+CHR$ 157+"A (ALAVA)?"
1640 DATA "QUE RIO PASA POR ELIZONDO?"
1650 DATA "EL RIO MUNDO ES UN AFLUENTE DEL..."
1660 DATA "QUE RIO DESEMBOCA EN CULLE-RA?"
1670 DATA "QUE RIO PASA POR SORIA?"
1680 DATA "EL RIO JALON ES UN AFLUENTE DEL..."
1690 DATA "EL RIO GENIL ES UN AFLUENTE DEL..."
1700 DATA "QUE RIO NACE ENTRE LAS SIE-RRAS DE ALCON Y CAZORLA?"
1710 DATA "QUE RIO PASA POR CORDOBA Y SEVILLA?"
1720 DATA "QUE RIO NACE EN FONTIBRE   (SANTANDER)?"
1730 DATA "QUE RIO PASA POR ZARAGOZA?"
1740 DATA "QUE RIO DESEMBOCA EN SANLU-CAR DE BARRAMEDA (HUELVA)?"
1750 DATA "EL RIO CADAGUA ES AFLUENTE DEL..."
1760 DATA "QUE RIO NACE EN LA PROVIN -CIA DE ASTURIAS?"
1770 DATA "QUE RIO PASA POR BILBAO?"
1780 DATA "QUE RIO NACE JUNTO AL CERRO DE SAN FELIPE EN CUENCA?"
1790 DATA "QUE RIO DESEMBOCA EN GUAR -DAMAR (ALICANTE)?"
1800 DATA "QUE RIO PASA POR VALENCIA?"
1810 DATA "QUE RIO NACE EN LA LAGUNA  DE FUENMI"+CHR$ 157+"A (LUGO)?
1820 DATA "EL RIO SIL ES UN AFLUENTE  DEL..."
1830 DATA "EL PISUERGA ES UN AFLUENTE DEL..."
1840 DATA "QUE RIO ES EL MAS LARGO DE LA PENINSULA (1.120 Km)?"
1850 DATA "QUE RIO NACE EN LOS PICOS  DE URBION?"
1860 DATA "QUE RIO NACE EN LA SIERRA  DE ALBARRACIN?"
```

```
1870 DATA "QUE RIO PASA POR LA CIUDAD DE BADAJOZ?"
1880 DATA "EL RIO GUADALBULLON ES UN  AFLUENTE DEL..."
1890 DATA "EL RIO CHANZA ES UN AFLUEN-TE DEL..."
1900 DATA "QUE RIO NACE EN FUENTE SE -GURA (JAEN)?"
1910 DATA "EL RIO TAIBILLA ES AFLUENTE DEL..."
1920 DATA "EL RIO HIJAR ES UN AFLUENTE DEL..."
1930 DATA "EL RIO GALLEGO ES AFLUENTE DEL..."
1940 DATA "QUE RIO DESEMBOCA  EN  LA  CIUDAD DE LISBOA?"
1950 DATA "EL RIO GUADIELA ES AFLUENTE DEL..."
1960 DATA "EL RIO ARAGON ES UN AFLUEN-TE DEL..."
1970 DATA "EL RIO IBAIZABAL ES AFLUEN-TE DEL..."
1980 DATA "QUE RIO NACE EN CASAS  DE  FUENTE GARCIA (TERUEL)?"
2030 REM
2040 REM **************************************
2050 REM * CREACION DE LA TABLA DE RESPUESTAS *
2060 REM **************************************
2061 REM
2070 RESTORE 2130
2080 FOR f=1 TO 54
2090    READ b
2100    LET t(f)=b
2110 NEXT f
2120 RETURN
2130 DATA 11,9,7,1,8,5,2,6,4,1,3
2140 DATA 2,2,1,3,11,10,9,11,8,7
2150 DATA 2,6,4,5,5,6,6,5,10,9
2160 DATA 10,7,8,7,1,1,2,3,2,3,4
2170 DATA 5,4,5,8,8,6,6,3,3,6,10
2180 DATA 3
2181 REM
2190 REM ***********************************************
2200 REM * FIN DE LA CREACION DE LA TABLA DE RESPUESTAS *
2210 REM ***********************************************
2230 REM
2240 REM **********************************
2250 REM * RUTINA DE CREACION DE VENTANAS *
2260 REM **********************************
2261 REM
2270 LET x1=3
2280 LET x2=28
2290 LET y1=2
2300 LET y2=16
2310 FOR f=x1+1 TO x2-1
2320    PRINT AT y1,f;CHR$ 143;AT y2,f;CHR$ 143
2330 NEXT f
2340 FOR f=y1 TO y2
2350    PRINT AT f,x1;CHR$ 133;AT f,x2;CHR$ 138
2360 NEXT f
2370 RETURN
```

TECNICAS DE ANALISIS



Diferentes tipos de códigos

Se pueden clasificar los códigos en dos grandes grupos desde el punto de vista semántico: códigos mnemónicos y códigos crípticos.

Denominamos mnemónicos a los códigos que aportan en los propios caracteres con que se forman y/o en la estructura del código, información sobre su significado. Con el neologismo «críptico» queremos significar que en los códigos así designados ni los signos que componen el código ni su situación aportan información alguna sobre el significado del código o la naturaleza del elemento a que se refieren.

a) *Códigos mnemónicos*

Estos códigos tienen la gran ventaja de que su utilización por las personas es más cómoda y segura (es más fácil de recordar, el propio código sugiere su aplicación, los errores se detectan más fácilmente, etc.). Tienen, por el contrario, la desventaja de que suelen ser más largos (para que el código realmente sugiera algo y no sea un crucigrama) y, como incorporan letras (e incluso en ocasiones símbolos especiales) su proceso por ordenador puede ser más complicado. Los códigos mnemónicos pueden ser de dos tipos:

— Con subcampos: son aquellos códigos organizados en diferentes campos, cada uno de los cuales tiene una significación diferente. Ejemplo de este tipo de códigos es el número de matrícula de los vehículos a motor en España; en ella hay tres campos: el primero, de una o dos letras, indica la provincia de matriculación (normalmente la inicial y, cuando hay ambigüedad entre dos provincias, otra letra característica); el segundo es un campo de cuatro dígitos con una numeración correlativa; el tercero, de dos letras, es una serie correlativa también. Normalmente las entregas efectuadas por una compañía suelen realizarse bajo un código de este tipo: una configuración de código típica es la siguiente:

99/99/99/9999/999

donde los tres primeros grupos de dos dígitos representan la fecha, el siguiente el número de pedido y el último el de cliente.

— De abreviatura: se llaman así a los códigos formados mediante la contracción del nombre completo que designa al elemento. Así, por ejemplo, se puede designar por «FICHMAS» al fichero maestro de una aplicación. Es usual utilizar un código mixto en que la primera parte es una abreviatura y va seguida de un número identificativo; por ejemplo, se pueden designar los diferentes ficheros de actualizaciones de una aplicación por los nombres «ACT1502», «ACT2302», «ACT0503», etc.

b) *Códigos crípticos*

Este tipo de código es más útil para su proceso por ordenador, pero más difícil de manejar por las personas. Se pueden considerar cuatro tipos diferentes de códigos crípticos:

— Secuencial: es la codificación más simple que se puede concebir: consistente en ir numerando, sucesivamente, los diferentes elementos del conjunto a codificar. Es muy sencilla de utilizar y permite muy fácilmente las extensiones de la serie de códigos ya establecidos, tiene el inconveniente de que no permite las inserciones y el problema (por lo demás común a todos los códigos crípticos) de que es difícilmente recordable y no aporta, por su estructura, ninguna facilidad para el procesamiento de los datos. Es usual añadir a este tipo de códigos algún o algunos caracteres (normalmente dígitos) de control; así suelen ser los números de las cuentas bancarias (aunque, en ocasiones, suelen tener estructura de subcampos).

— Secuencial, con incrementos mayores que la unidad. Se construye este tipo de códigos cuando es previsible una actividad grande de inserción de nuevos códigos, pero la identificación de cada elemento mediante el código no es interesante en absoluto. Esta es la situación típica del número de secuencia que se pone en las líneas de programa: es usual numerarlas de diez en diez unidades (0010, 0020, 0030..., 0090, 0100, etc.); de este modo, las posteriores actualizaciones van recibiendo números intermedios (en el ejemplo anterior, 0013 y 0016, por ejemplo, en una primera actualización; 0011, 0017 y 0019 en una segunda, etc.

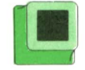
Naturalmente, el proceso de intercalación nunca puede ser indefinido, aunque si se prevé una actividad de inserción grande, suele utilizarse un módulo de incremento de la secuencia de intercalación mayor (numerar de cien en cien o de mil en mil).

— Secuencial por grupos. Se numeran secuencialmente los elementos, pero habiéndolos separado previamente en grupos. Por ejemplo, se reservan los primeros cien números para los diferentes modelos de ordenadores disponibles (001 Spectrum, 002 Spectrum Plus, 003 Commodore 64, 004 MSX, etc.), los siguientes cien para los periféricos (100 Impresora Epson, 101 C. Itho, 102 bmc, etc.), otros cien sucesivos para los paquetes de software (200 Contabilidad, 201 Facturación, etc.). Este tipo de código aporta alguna información sobre el elemento representado pero sigue exigiendo alguna tabla de conversión y referencia.

— Por subcódigos estructurados. En ocasiones, se define una estructura de subcampos y subcódigos, pero de tal modo que, dentro de cada subcódigo, la numeración sea secuencial. Por ejemplo, el código de series que acaba de ser presentado podía estructurarse del siguiente modo:

9/99/99/9/999

tomando el primer dígito para representar la línea de productos (1 ordenadores domésticos, 2 ordenadores personales, 3 miniordenadores, etc.), los dos siguientes para la marca (01 Sinclair, 02 Sony, 03 Philips, 04 Commodore, etc., dentro de la línea de productos 1; por supuesto, suele volver a aparecer Philips, o Sony en la línea 2, y así con las restantes), otros dos para el modelo (dentro de la línea 1 y de la marca 01, 01 puede ser el modelo Spectrum 48K, 02 el Spectrum Plus, etc., 99 todos los modelos —código útil cuando un paquete de software o un periférico sirve para todos los modelos de un tipo—, uno más para el tipo de producto dentro de cada marca (1 para el propio ordenador, 2 para periféricos, 3 para software, etc.), y, por fin, tres dígitos para identificar a cada elemento del subconjunto de que se trate.

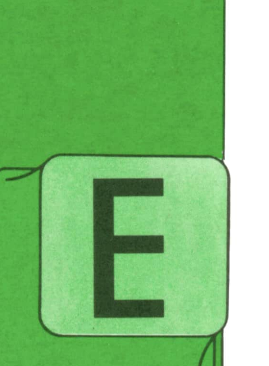


Utilización de los códigos

Es importante tener en cuenta, asimismo, las posibles utilizaciones de un código en el momento de su elección y, según sus características, disponer (o no) información de ayuda para quien debe escribir el código (incluir la tabla de valores posibles), estudiar su posición en un impreso según la complejidad del sistema elegido, prever controles adecuados durante el proceso de depuración de los datos si los códigos son poco (o nada) mnemotécnicos, etc.

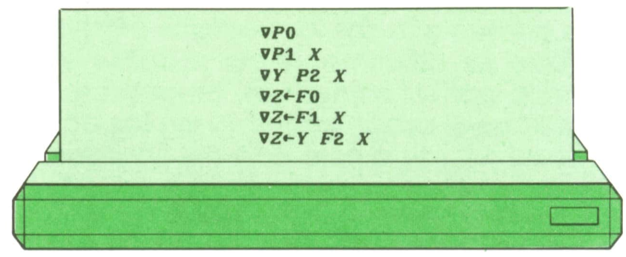


Programación modular (continuación)



EN APL, un procedimiento se define exactamente igual que una función, con la única diferencia de que no existe resultado y, por tanto, no aparece en la línea de cabecera un nombre de variable separado de la función por una flecha de asignación. Además, al igual que en PASCAL, un procedimiento se invoca sin necesidad de utilizar palabras reservadas, simplemente mencionando su nombre.

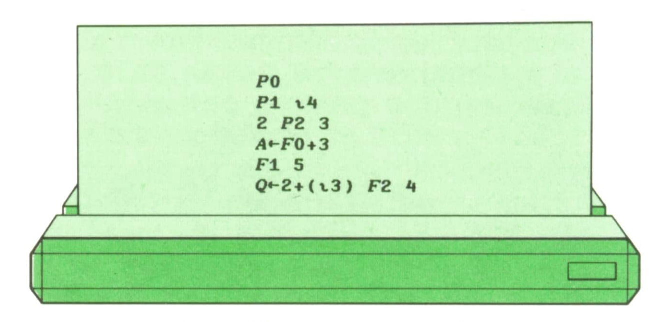
En APL, los argumentos de funciones y procedimientos están limitados a un máximo de dos. Además, no se expresan obligatoriamente entre paréntesis, a la derecha del nombre del módulo o subrutina, sino a derecha e izquierda. Es posible, sin embargo, definir tanto procedimientos como funciones con un solo argumento o con ninguno (recuérdese la función MEDIA, del capítulo anterior). Todo esto significa que en este lenguaje existen seis formas diferentes de subprogramas: funciones o procedimientos (es decir, módulos con o sin resultado) con cero, uno o dos argumentos. Veamos qué forma tomarían sus cabeceras, utilizando los nombres X e Y para los argumentos, el nombre Z para el resultado, y nombres que comienzan por F para las funciones y por P para los procedimientos. (Naturalmente, podríamos haber utilizado cualquier otro nombre para todas estas variables y subprogramas).



Los ejemplos anteriores representan las cabeceras en el siguiente orden:

1. Procedimiento sin argumentos.
2. Procedimiento con un argumento.
3. Procedimiento con dos argumentos.
4. Función sin argumentos.
5. Función con un argumento.
6. Función con dos argumentos.

La forma de invocar los subprogramas anteriores es semejante a la que presentamos en los ejemplos siguientes:



cuya explicación es como sigue:

1. P0 es un procedimiento sin argumentos ni resultado. Por tanto, la única forma posible de invocarlo es mediante una línea donde aparezca su nombre totalmente aislado.

2. P1 tiene un argumento derecho, que puede ser sustituido por cualquier expresión (en nuestro ejemplo, la serie de números del 1 al 4). Pero, como no tie-

ne resultado, no se puede operar con él, por lo que su nombre deberá aparecer en el extremo izquierdo de la línea.

3. P2 es un procedimiento con dos argumentos, uno de los cuales debe escribirse a su izquierda y el otro a su derecha. En nuestro ejemplo, estos argumentos son, respectivamente, los números 2 y 3. Cualquiera de ellos puede sustituirse por una expresión, pero en tal caso, el de la izquierda al menos debe escribirse entre paréntesis (los paréntesis a la derecha son optativos).

4. F0 es una función sin argumentos. Por tanto, se la invoca únicamente con su nombre, pero como tiene resultado puede entrar a formar parte de una expresión, como la del ejemplo. Lo que hace esta línea es obtener el valor generado por F0, sumarle 3 y asignar el resultado a la variable A.

5. F1 es una función con un argumento, que debe colocarse siempre a su derecha. En nuestro caso, se trata del valor 5. Como la línea no se prolonga a la izquierda de F1, y ésta tiene resultado explícito, dicho resultado se escribirá automáticamente en la pantalla. (Recuérdese que, en APL, cualquier línea que no termine en una asignación o una transferencia y tenga resultado explícito equivale a una orden de que dicho resultado se escriba en la pantalla.)

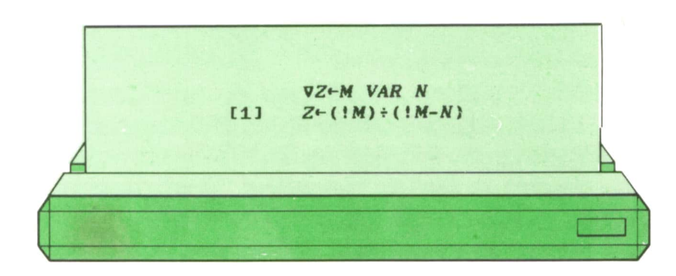
6. Finalmente, F2 es una función con dos argumentos. El derecho será el número 2 y el izquierdo la serie de números de 1 a 3. (Obsérvese que, al igual que en los procedimientos, el argumento izquierdo debe siempre colocarse entre paréntesis, si se trata de una expresión y no de un valor aislado.) Al resultado de la función se le suma 2 y el valor final queda asignado a la variable G.

Hemos visto que los argumentos y el resultado de las subrutinas APL son siempre variables locales. Es decir, su valor no entra en conflicto con otros que pueda tomar una variable con el mismo nombre fuera del módulo, aunque este valor será inaccesible dentro de él. Además, podemos incluir otras variables locales, a nuestro gusto. Hay dos formas principales de hacerlo:

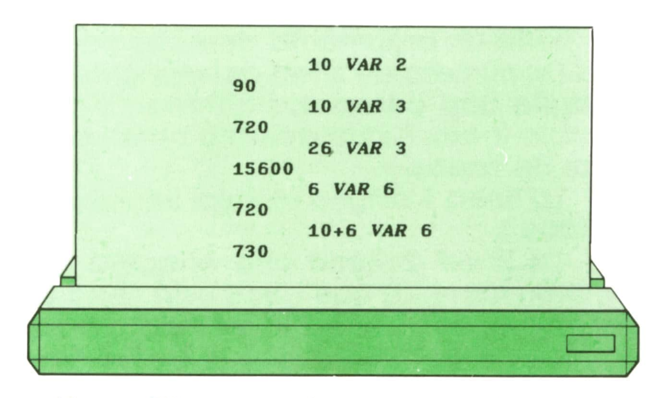
— Todas las etiquetas definidas dentro de una subrutina APL se consideran automáticamente como variables locales, cuyo valor es igual al número de la línea en que están definidas. No es posible asignar otros valores a estas variables.

— Todos los nombres que aparezcan en la cabecera de la subrutina, después del argumento derecho (si existe) o del nombre del módulo (si no tiene argumentos), separados por puntos y comas, se considerarán variables locales, que inicialmente no tendrán valor alguno.

Veamos algunos ejemplos de funciones y procedimientos APL:


```
      ∇Z←M VAR N
[1]   Z←(!M)÷(!M-N)
```

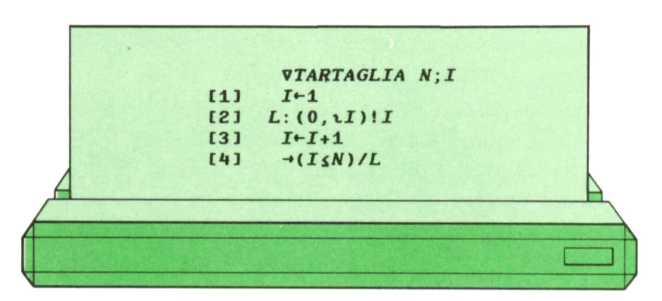
Esta función de dos argumentos calcula el número de variaciones sin repetición que se pueden formar con M elementos tomados de N en N, donde nos importa el orden en que aparezcan (es decir, 1 2 3 4 es una variación distinta de 4 3 2 1) aplicando la conocida fórmula que calcula dicho número como el cociente entre el factorial de M y el factorial de M − N. En APL, el signo de admiración colocado a la izquierda de un número, calcula directamente el factorial de dicho número. Veamos algunos ejemplos del uso de esta función:


```
      10 VAR 2
90
      10 VAR 3
720
      26 VAR 3
15600
      6 VAR 6
720
      10+6 VAR 6
730
```

Si se utiliza con dos argumentos, uno a su derecha y el otro a su izquierda (N!M), el signo de admiración calcula en APL el número de combinaciones sin repetición que se pueden formar con M elementos en grupos de N, sin importarnos el orden de los elementos dentro de un mismo grupo. Veamos algunos ejemplos:

```
        2!5
10
        3!5
10
        4!5
5
        0!5
1
        2 3 4 0!5
10 10 5 1
        0 1 2 3 4 5!5
1 5 10 10 5 1
        (0,⍳5)!5
1 5 10 10 5 1
```

Se observará que esta operación puede aplicarse lo mismo a números aislados como a series de ellos. Vamos a utilizarla para hacer un procedimiento que nos permita construir el triángulo de Tartaglia, que nos genera los números combinatorios en filas sucesivas:



```
          ∇TARTAGLIA N;I
[1]       I←1
[2]     L:(0,⍳I)!I
[3]       I←I+1
[4]       →(I≤N)/L
```

Veamos cómo funciona este procedimiento:

— La cabecera nos dice que el nombre del procedimiento es TARTAGLIA, y que tiene un argumento derecho llamado N (el número de línea del triángulo de Tartaglia que deseamos obtener) y una variable local, I, que actuará como contador de líneas.

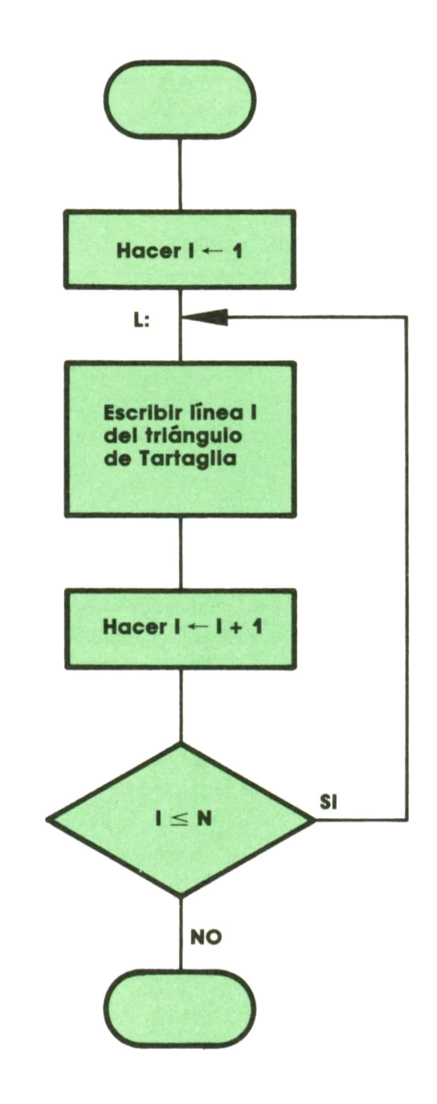
— La línea 1 asigna el valor inicial a la variable I.

— La línea 2 tiene una etiqueta (L), también local. Lo que hace esta línea es generar (y escribir en el terminal), pues no termina en asignación ni en transferencia) una línea entera del triángulo de Tartaglia, produciendo los números combinatorios de I elementos tomados de todas las formas posibles (desde 0 en 0 hasta I en I).

— La línea 3 incrementa el valor del contador.

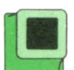
La línea 4 es una transferencia condicional a la instrucción de etiqueta L si el valor de I es menor o igual que el del argumento N.

Veamos el organigrama de esta función:



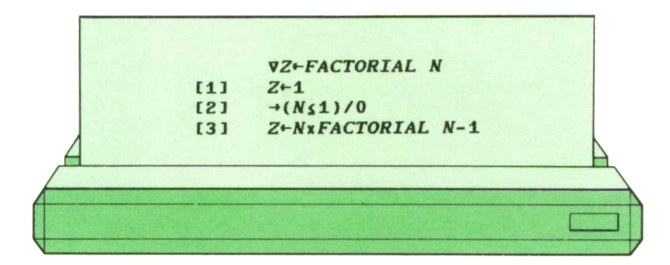
y un ejemplo de su utilización:

```
      TARTAGLIA 10
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
```

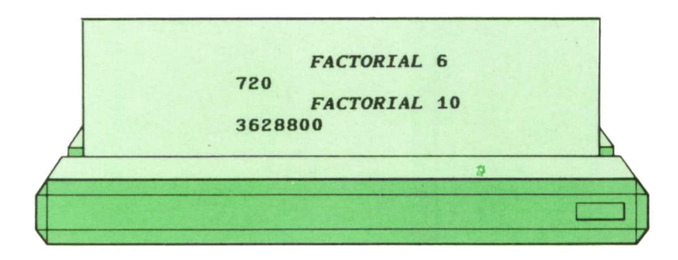


Funciones recursivas

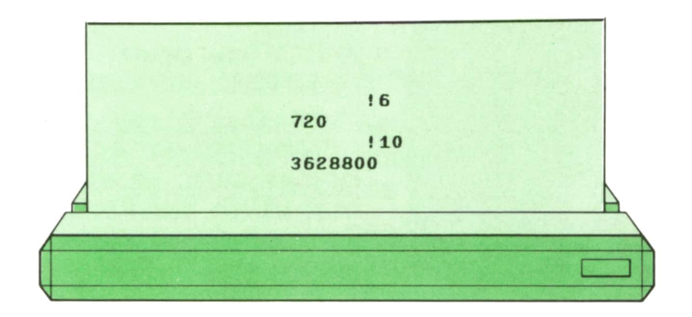
APL permite que un módulo sea invocado dentro de su propia definición. Cuando esto ocurre, se dice que el módulo es recursivo. La posibilidad de construir funciones y procedimientos recursivos proporciona una enorme potencia a un lenguaje de programación. Veamos un ejemplo sencillo de función recursiva:



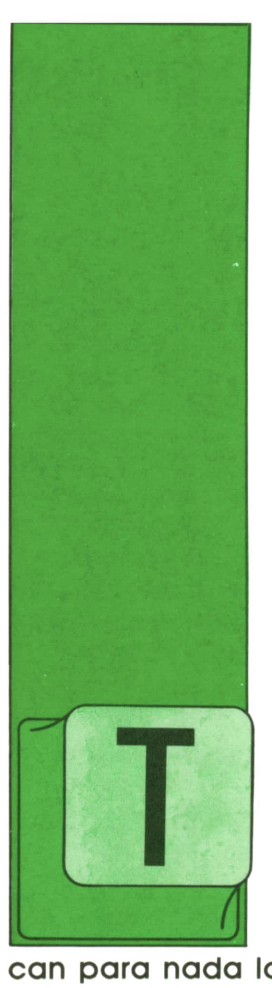
Veamos cómo funciona: se trata de una función con un solo argumento (el número cuyo factorial queremos calcular). La línea 1 inicializa a 1 el valor del resultado. Este es el resultado correcto si N era 0 ó 1. La línea 2 lo comprueba y, en caso afirmativo, da por terminada la ejecución de la función (la transferencia a la línea cero significa, en APL, terminar la ejecución del módulo). Finalmente, la línea 2 calcula el factorial de N como el producto de N por el factorial de N – 1. El factorial tiene, en efecto, esta propiedad. Veamos un ejemplo de su uso:



Esta función sirve únicamente como ejemplo sencillo de recursividad. En la práctica no se usa, pues, como ya hemos dicho, en APL se puede calcular directamente la función factorial con el signo de admiración:



LOGO





Operaciones básicas con números

TANTO las operaciones básicas que vamos a ver ahora como otras especialidades que veremos más tarde, son funciones, es decir, son órdenes que no modifican para nada las características de la tortuga, sino que hacen que ésta nos devuelva un resultado. Al igual que en otras ocasiones, con este resultado tenemos que hacer algo. Lo más sencillo es escribirlo, pero, en general, el resultado de una operación con números los utilizaremos con cualquier comando con el que se pueda poner un número.

Como es lógico, las cuatro operaciones básicas con números son la suma, resta, multiplicación y división. La forma general de especificarlas es:

n1 op n2

siendo *n1* el primer número o primer operando, *n2* el segundo número o segundo operando y *op* el operador u operación:

+	suma
–	resta
*	multiplicación
/	división

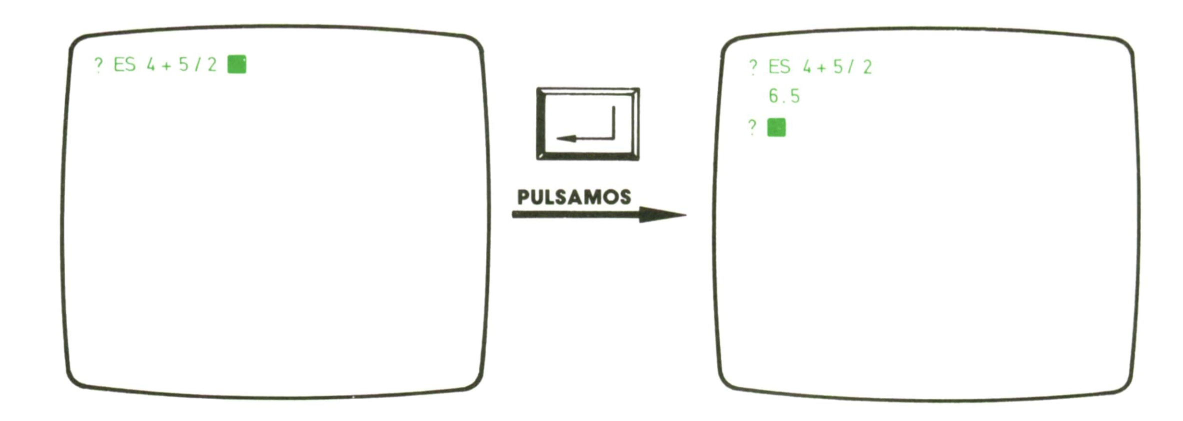
Se pueden encadenar varias operaciones teniendo en cuenta que existe una prioridad entre estos operadores, es decir, que la tortuga calcula los resultados siguiendo un determinado orden:

1. Divisiones.
2. Multiplicaciones.
3. Restas.
4. Sumas.

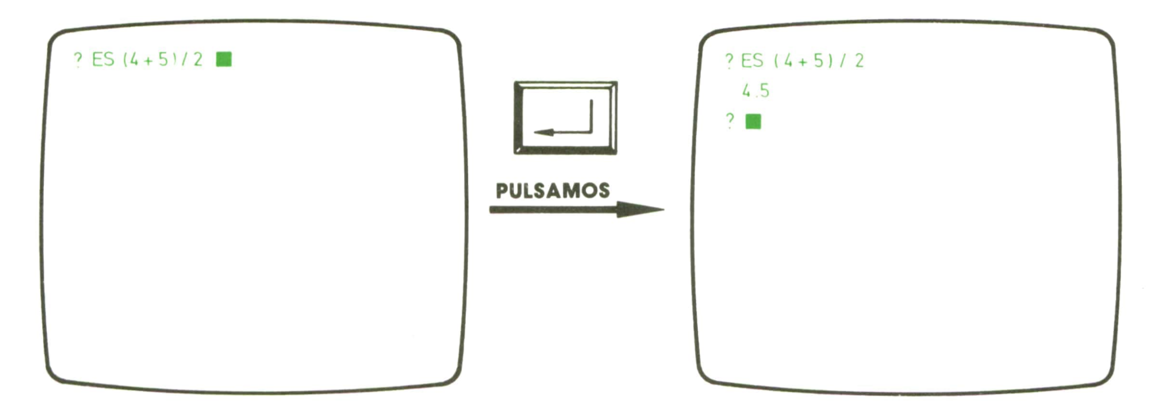
excepto cuando nosotros le indicamos que lo cambie mediante la utilización de paréntesis.

Vemos la diferencia del resultado de una operación usando y sin usar paréntesis:

— *Sin paréntesis*



— *Con paréntesis*



Otra manera de escribir las operaciones básicas

Algunas de las operaciones básicas con números se pueden especificar mediante un nombre en lugar de utilizar el operador correspondiente. En este caso, la forma de escribirlas es así:

op n1 n2

siendo **n1** el primer número, **n2** el segundo número y **op** el nombre de la operación:

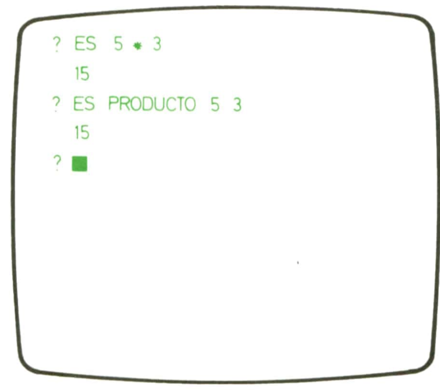
SUMA	suma
PRODUCTO	multiplicación
DIV	división

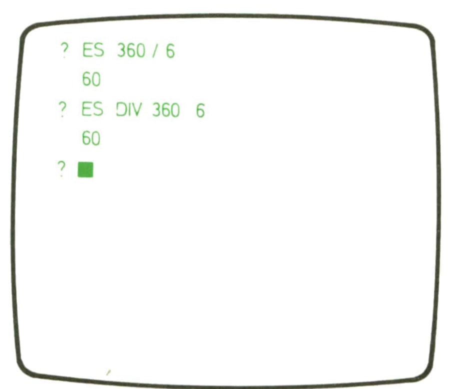
Por tanto, tenemos que:

— **SUMA** es equivalente a +
— **PRODUCTO** es equivalente a *
— **DIV** es equivalente a /

como podemos ver en estos ejemplos:

```
? ES 8 + 4
12
? ES SUMA 8 4
12
?
```





A la hora de utilizar una de las dos formas de escribir una misma operación hay que tener un poco de cuidado porque el orden en que se ponen los números y la operación es diferente.

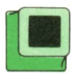
Os proponemos

1. Teniendo en cuenta el orden en el cual realiza la tortuga las operaciones, intenta averiguar qué resultado escribiría en cada caso:

? ES 4 + 3 − 2
? ES 5 * 4 + 1
? ES 17 − 6 * 2
? ES (17 − 6) * 2
? ES 3 * 8/4
? ES 3 + 12/3
? ES (3 + 12)/3

2. Escribe estas operaciones de otra forma si es posible:

4 * 3	PRODUCTO 4 3
5 / 7	
2 + 7	
7 − 2	
	SUMA 5 3
	DIV 4 8
	PRODUCTO 1 6



Otras operaciones con números

Nuestra amiga es capaz de realizar otras operaciones que no se utilizan con tanta frecuencia como las anteriores pero que pueden ser interesantes en algunos casos. Veamos algunas de ellas.

La función

COCIENTE n1 n2

devuelve el cociente de la división del número **n1** por el número **n2,** mientras que la función

RESTO n1 n2

devuelve el resto de dividir el número **n1** por el número **n2.**

Así tendríamos que:

```
? ES 5 / 2
  2.5
? ES DIV 5 2
  2.5
? ES COCIENTE 5 2
  2
? ES RESTO 5 2
  1
? ■
```

La función

ENTERO n

devuelve la parte entera del número **n,** es decir, es equivalente a restar a un número su parte decimal.

Por otro lado, la función

REDONDEA n

devuelve el número, que es el resultado de redondear el número **n** al entero más próximo.

Veamos la diferencial con algunos ejemplos:

```
? ES ENTERO 0.5
  0
? ES REDONDEA 0.5
  1
? ES ENTERO 3.47
  3
? ES REDONDEA 3.47
  3
? ■
```

```
? ES ENTERO 12.68
  12
? ES REDONDEA 12.68
  13
? ES ENTERO -7
  -7
? ES REDONDEA -7
  -7
? ■
```



Algunos procedimientos con números

Ya hemos visto que la tortuga nos da la posibilidad de hacer sumas, multiplicaciones y divisiones de dos maneras: con un operador o con un nombre de función. Sin embargo, para la resta sólo podemos utilizar un operador. Vamos, pues, a definir un procedimiento que nos permita hacer también la resta usando un nombre.

Una manera sencilla de hacerlo sería:

```
? PARA RESTA :NUM1 :NUM2
> ES :NUM1 – :NUM2
> FIN
```

Otra forma sería teniendo en cuenta que el restar un número de otro es equivalente a quitarle al primero tantos unos (1) como valga el segundo.

Por ejemplo:

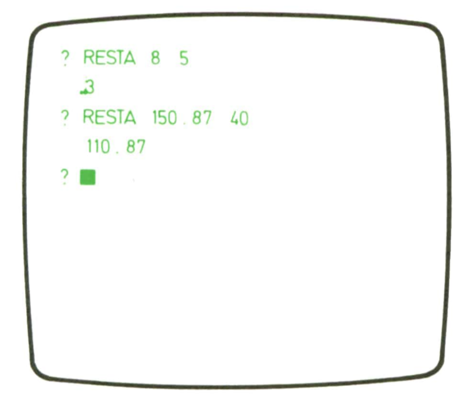
5 − 3 = 5 − 1 − 1 − 1
tres unos

Con esto, podemos escribir:

```
?  PARA RESTA :NUM1 :NUM2
>  REPITE :NUM2 (HAZ "NUM1 :NUM1 −1)
>  ES :NUM1
>  FIN
```

es decir, vamos decrementando el valor del primer número de uno en uno y lo repetimos tantas veces como nos indique el valor del segundo número.

Probemos a ver si funciona:



Ahora, supongamos que queremos escribir los cinco primeros números. Pondríamos:

```
?  PARA CUENTA1
>  ES 1
>  ES 2
>  ES 3
>  ES 4
>  ES 5
>  FIN
```

Si no queremos empezar siempre por el 1, necesitamos una variable para decirle a la tortuga el primer número. Entonces, nos quedaría:

```
?  PARA CUENTA2 :NUM
>  REPITE 5 (ES :NUM HAZ "NUM :NUM + 1)
>  FIN
```

Si en lugar de escribir cinco números quisiéramos poder decir la cantidad de números, utilizaríamos otra variable:

```
?  PARA CUENTA3 :NUM :CANT
>  REPITE :CANT (ES :NUM HAZ "NUM :NUM
   + 1)
>  FIN
```

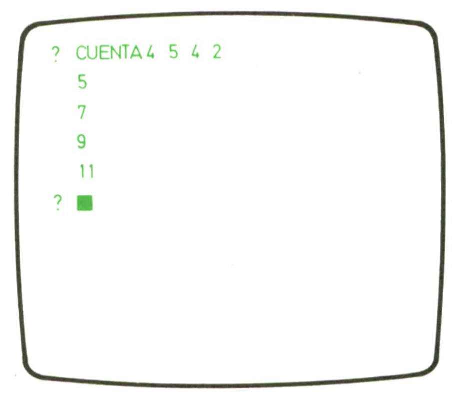
Por último, si en vez de ir incrementando los números de 1 en 1 queremos escribirlos de 2 en 2 o de 5 en 5, añadiríamos una tercera variable:

```
?  PARA CUENTA4 :NUM :CANT :INC
>  REPITE :CANT (ES :NUM HAZ "NUM :NUM
   + :INC)
>  FIN
```

Así, al escribir:

```
?  CUENTA4 5 4 2
```

nos quedaría:



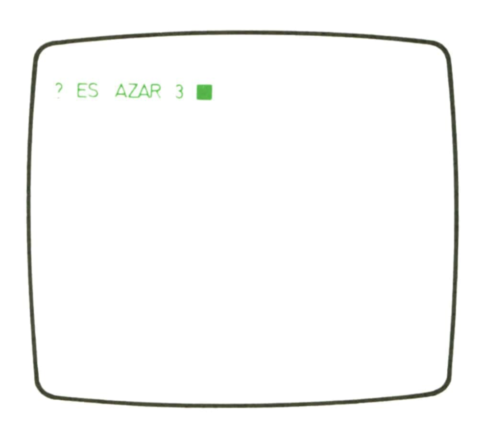
Una función un poco especial

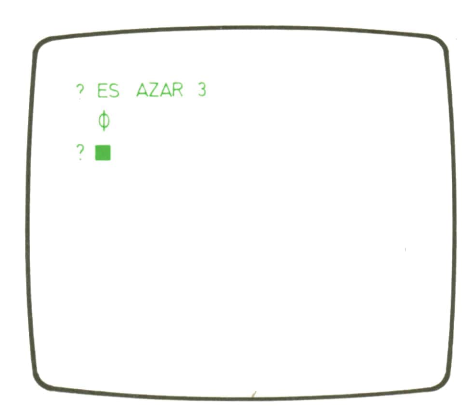
Se dice que una cosa es *aleatoria* cuando no podemos saber con exactitud el resultado que nos va a dar. Por ejemplo, cuando jugamos al parchís y tiramos el dado, sabemos que nos va a salir un valor entre 1 y 6, pero no podemos decir exactamente cuál va a ser.

Pues bien, la función

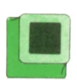
AZAR n

hace una cosa parecida. Con ella, la tortuga nos devuelve un número aleatorio que va a estar comprendido entre **0** y *n −1*.





Por ejemplo, si ponemos
podemos obtener como resultado 0, 1 ó 2.

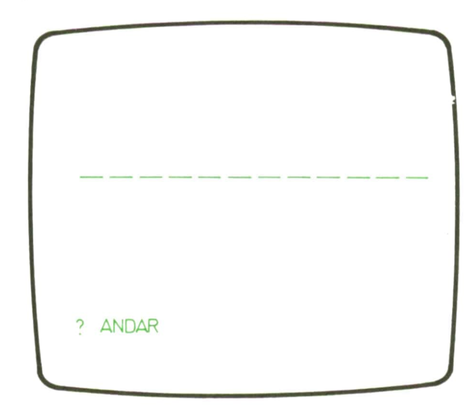


Utilización de la función AZAR

Supongamos que queremos que la tortuga vaya avanzando de izquierda a derecha por la pantalla un número aleatorio de pasos de forma discontinua, es decir, alternando que la tortuga pinte y no pinte. Escribiríamos:

```
? PARA ANDAR
> SL PONPOS (-100 0) BL
> PONRUMBO 90
> REPITE 20 (AV AZAR 5 SL AV AZAR 5 BL)
> OT
> FIN
```

Al ejecutarlo, nos quedaría algo así:



Si ahora queremos que la tortuga vaya avanzando un número de pasos aleatorio y que también gire un número de grados entre 0 y 90, tenemos que poner:

```
? PARA AVANGIRA
> SL PONPOS (-100 0) BL
> REPITE 25 (AV AZAR 10 GD AZAR 91)
> OT
> FIN
```

Al ejecutar este procedimiento, la tortuga se mueve y gira aleatoriamente por la pantalla.

Por último, vamos a suponer que tenemos definido un procedimiento que dibuja polígonos regulares:

```
? PARA POLI :LADO :NUMLADOS
> REPITE :NUMLADOS (AV :LADO GD 360 /
  :NUMLADOS)
> OT
> FIN
```

Si queremos que cada vez que la tortuga dibuje un polígono, el color del fondo de la pantalla y el de la figura sean aleatorios, debemos usar la función AZAR teniendo en cuenta que la tortuga dispone de 16 colores distintos.

```
? PARA POLI :LADO :NUMLADOS
> PONFONDO AZAR 16
> PONCL AZAR 16
> REPITE :NUMLADOS (AV :LADO GD 360 /
  :NUMLADOS)
> OT
> FIN
```

Puede darse el caso de que al ejecutar este procedimiento alguna vez no veamos nada en la pantalla: Esto es porque puede dar la casualidad de que al obtener el número para el color del fondo y para el color del lápiz, la tortuga saque el mismo.



Os proponemos

1. Escribe un procedimiento que se llame SUMAR y que obtenga la suma de dos números de forma parecida al procedimiento RESTA.

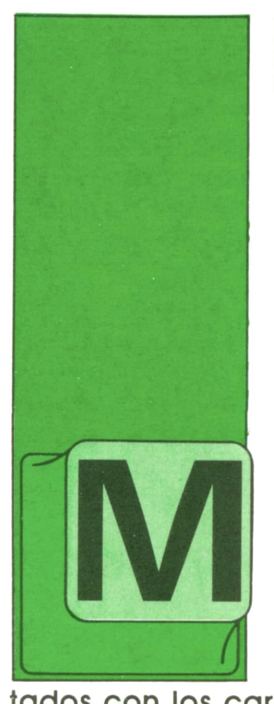
2. Define un procedimiento parecido al CUENTA4, pero que en lugar de escribir los números en orden ascendente lo haga en orden descendente.

3. Suponiendo que tenemos definido el procedimiento FLOR que dibuja 8 heptágonos girados, añádele lo necesario para que cada uno sea de un color aleatorio.

```
? PARA FLOR
> REPITE 8 (REPITE 7 (AV 50 GD 360 / 7) GD
  45)
> OT
> FIN
```

4. Piensa cómo dibujarías una circunferencia de tamaño aleatorio.

PASCAL





Ficheros de texto

MUY frecuentemente lo que se desea guardar en un fichero son datos tal como aparecerían en la pantalla del ordenador, o sea, frases, palabras o números representados con los caracteres que les corresponden. Se suele guardar así información que en algún momento ha de ser leída tal cual, sin ser sometida a ningún proceso. Una carta, un capítulo de este libro o el texto de un programa PASCAL son casos típicos.

Para ello habría que utilizar estructuras de tipo FILE OF CHAR que permitirían guardar los datos en forma de chorro de caracteres.

Los textos normalmente se encuentran divididos en renglones, siendo lo más corriente al guardarlos en ficheros utilizar unos caracteres especiales para indicar cuándo acaba un renglón y empieza el siguiente. Con los códigos ASCII el fin de una línea se indica a menudo con la pareja formada por los caracteres cuyos ordinales son 13 y 10, que se denominan, respectivamente, "retorno de carro" y "salto de línea" por su significado para teletipos e impresoras.

Utilizando el ejemplo de una cinta de magnetófono, es como si, habiendo grabado un libro en ella, se hubiera indicado el final de cada renglón con un silbido o una palmada.

Tan habituales son estos ficheros que su tipo se encuentra ya predefinido con el nombre TEXT.

Como tales ficheros de caracteres, se podrían utilizar todos los procedimientos generales vistos hasta ahora para leer o escribir de carácter en cárácter; sin embargo, el PASCAL permite hacer un tratamiento un poco especial de los ficheros de tipo TEXT para manejar cómodamente la cuestión de las líneas, por medio de los procedimientos READ, READLN, WRITE y WRITELN junto con la función EOLN. Si definimos:

var T: text;

T sería una variable con un fichero de texto asociado, y podríamos hacer lo siguiente con ella:

eoln (T) Esta función (End Of LiNe, fin de línea) devuelve TRUE si nos encontramos al final de una línea y FALSE en caso contrario.

readln (T) Cuando se está leyendo el fichero asociado a T, con esto nos posicionaríamos en el primer carácter de la siguiente línea a aquélla en que nos encontramos.

writeln (T) Al escribir en el fichero asociado, éstos pondrián la marca de fin de línea justo a continuación del último carácter escrito, quedando todo listo para comenzar a escribir los de la siguiente línea.

Por otra parte, con los ficheros de texto podemos utilizar los procedimientos WRITE y READ de la siguiente manera:

read (T,C)
equivale a
begin C:= T^; get (T) end
write (T,C)
equivale a
begin T^:= C; put (T) end

o sea, leer (o escribir) el carácter C en la posición del fichero T en que nos encontramos, para pasar después a la siguiente posición.

Sin embargo, la utilización de READ y WRITE va más allá: es posible leer o escribir varios datos con una sola instrucción:

read (T,C1,C2) equivale a **begin read (T,C1); read (T,C2) end**

y análogamente con WRITE. Si tras una instrucción READ o WRITE, respectivamente, hubiera que ejecutar READLN o WRITELN, se podría utilizar una sola instrucción:

writeln (T,C) equivale a **begin write (T,C); writeln (T) end**

Esto último, por ejemplo, supondría escribir el carácter C seguido de una marca de fin de línea.

Además, no sólo se pueden leer o escribir caracteres: es posible escribir valores de tipo INTEGER o REAL utilizando incluso definiciones de espaciado, de manera que lo que se escriba sea su representación con caracteres. Si, por ejemplo, pusiésemos:

```
write (T,N:3)
```

y N fuera una variable INTEGER con valor 54, esto sería equivalente a poner:

```
begin
  write (T,' ');
  write (T,'5');
  write (T,'4');
end
```

También se pueden escribir en un fichero valores de tipo BOOLEAN, lo que equivaldría a escribir la palabra 'FALSE' o 'TRUE', y, normalmente, strings y datos de tipo array de caracteres.

Igualmente, si el fichero contuviera, por ejemplo, los cuatro caracteres '36' y estuviésemos posicionados sobre el primero de ellos (un espacio en blanco) o el segundo, la ejecución de read(T,N) o de readln(T,N) haría que la variable entera N tomara el valor 36, quedándonos después posicionados en el carácter siguiente al seis o al comienzo de la siguiente línea, según el caso.

En otras palabras, con READ, READLN, WRITE y WRITELN se puede hacer EXACTAMENTE lo mismo que hacíamos para escribir datos en pantalla o leerlos de teclado, sólo que enviando los caracteres que normalmente aparecerían en la pantalla a un fichero, y recogiendo los datos que normalmente se leerían del teclado de un fichero. Al leer los datos de un fichero, la marca de fin de línea sería equivalente a pulsar la tecla de RETURN (intro, newline, etc.) cuando se lee del teclado.

Esta coincidencia no es casual. La presentación en pantalla toma la estructura de una secuencia de caracteres distribuida en renglones, y los datos que se introducen desde teclado son también secuencias de caracteres separadas entre sí a golpes de RETURN. Por ello, en PASCAL se trata al dispositivo de presentación de datos (normalmente la pantalla) y al de recogida de datos (normalmente el teclado) como si fueran ficheros.

Estos ficheros se encuentran predefinidos y se llaman respectivamente, OUTPUT (salida en inglés), e INPUT (entrada). Los procedimientos RESET y REWRITE no se utilizan, como es lógico, con estos ficheros.

Para escribir un dato en el fichero OUTPUT y leer otro del fichero INPUT, haríamos:

```
write (output,N);
read (input,A);
```

Cuando en un procedimiento de los existentes para ficheros no aparece como primer parámetro el nombre de una variable de tipo FILE, el PASCAL supone que el fichero que falta es, según el caso, OUTPUT o INPUT, por lo que lo anterior equivale a:

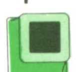
```
write (N);
read (A);
```

En otras palabras: todos los ejemplos del libro en que hemos utilizado esos procedimientos han sido casos particulares de escritura o lectura de ficheros de tipo TEXT. En todos esos ejemplos podríamos, por tanto, haber enviado datos a un fichero en lugar de a la pantalla para su posterior utilización, o leído los datos de un fichero en lugar del teclado, sin más que colocar en cada instrucción de entrada o salida el nombre de la variable de tipo FILE asociada al fichero (y hacer los preparativos previos para manejar ficheros que precise nuestro ordenador).

Para enviar o recoger datos de otros dispositivos, el método suele ser el mismo. Por ejemplo, es corriente que, en caso de haber impresora, ésta figure como un fichero con un nombre predefinido; este fichero sería similar al fichero OUTPUT.

La posibilidad de salvar y recoger números de ficheros utilizando, no los códigos que internamente emplea el ordenador, sino su representación por medio de caracteres, tal como los escribiría un ser humano, sirve para que datos preparados y salvados por un programa en un orde-

nador dado se puedan leer con otro programa preparado con un compilador distinto o con un ordenador de distinto tipo.



Ejemplos

Como ya mencionamos en su momento, el compilador que vamos a utilizar para todos los ejemplos en que sea preciso escoger uno determinado será el Turbo Pascal.

Con este compilador, para asociar un fichero de disco a una variable de tipo FILE, hay que ejecutar el procedimiento ASSIGN:

assign (F,'c:\Godos.lst');

esto asociaría a la variable F con el fichero Godos.Lst que se encuentra (o se va a encontrar, en caso de crearse ahora) en el directorio raíz de la unidad de disco C.

El nombre del fichero podría estar especificado con una variable o expresión en lugar de con una constante entre apóstrofos. La llamada a este procedimiento ha de ser el primer paso a ejecutar para trabajar con un fichero.

Cuando ya se ha terminado de trabajar con un fichero, hay que indicárselo al sistema operativo para que, en su caso, haga efectivos los cambios introducidos, por medio del procedimiento CLOSE:

close (F);

Explicadas sus peculiaridades, podemos comenzar ya con los ejemplos.

En primer lugar, vamos a escribir un programa para guardar en un fichero de texto los datos que tecleemos. Los datos tecleados para cada línea se recogerán en un variable ARRAY OF CHAR, pero en el fichero sólo se guardarán los caracteres realmente tecleados. (Cuando el número de éstos es menor que los que tiene definidos la variable, al leerse ésta se completa automáticamente con espacios en blanco).

```
program Guardar;
(* Este programa precisa de algunos cambios para correr *)
(* con un compilador distinto del Turbo Pascal.         *)

const
 MaxLong = 80;

type
 Renglon_t = array [1..MaxLong] of char;

var
 Fichero: text;
 Nombre : array [1..20] of char;     (* para el nombre del fichero *)
 Renglon: Renglon_t;              (* para leer de teclado cada línea *)
 I,L    : integer;

(*-------------------------------------------------------------------*)
function  Long (T: Renglon_t): integer;

(* Devuelve la longitud real del texto T.  Se explora el texto de *)
(* atrás para adelante hasta encontrar el primer carácter válido: *)

  var I: integer; Parar: boolean;
 begin
  I:= MaxLong;
  Parar:= false;
  while (I > 0) and not Parar do
    if (T[I] <> ' ') and (T[I] <> chr(0)) then Parar:= true
                                          else I     := I-1;
  Long:= I
 end;

(*-------------------------------------------------------------------*)
begin
 write ('Nombre del fichero: '); readln (Nombre);
 assign (Fichero,Nombre);
 rewrite (Fichero);
 writeln ('Empiece a escribir el texto, separando las líneas ',
          'con INTRO.');
 writeln ('Para acabar, pulse INTRO dos veces.');
 writeln;

 repeat
   readln (Renglon);                        (* lee renglón de teclado *)
   L:= Long (Renglon);
   if  L > 0 then      (* si hay caracteres, guardalos en Fichero *)
```

```
      begin
       for I:= 1 to L do write (Fichero, Renglon [I]);
       writeln (Fichero)        (* tras cada línea, marcar el final *)
      end
   until L = 0;

   close (Fichero)
end.
```

Cuando, tras escribir una línea, pulsamos Intro, la línea se guarda en el fichero y el programa se vuelve a parar en la instrucción READLN; si a continuación volvemos a pulsar Intro sin haber escrito nada, Renglon no contendrá más que espacios en blanco (o chr(0) con otros compiladores), por lo que Long devolverá el valor 0. Se utiliza esta circunstancia para acabar la ejecución del programa.

Este programa serviría, por ejemplo, para escribir programas PASCAL línea a línea y guardarlos en un fichero. A diferencia de un buen programa editor, no es posible corregir los errores una vez se ha pasado a la siguiente línea.

Por último, vamos a escribir un programa para presentar por pantalla el contenido de un fichero de texto.

```
program Recoger;
(* Este programa precisa de algunos cambios para correr *)
(* con un compilador distinto del Turbo Pascal.         *)

var
 Nombre : array [1..20] of char; (* Para el nombre del fichero *)
 Fichero: text;
 C       : char;

begin
 write ('Nombre del fichero: '); readln (Nombre);
 assign (Fichero,Nombre);
 reset (Fichero);
 ClrScr;

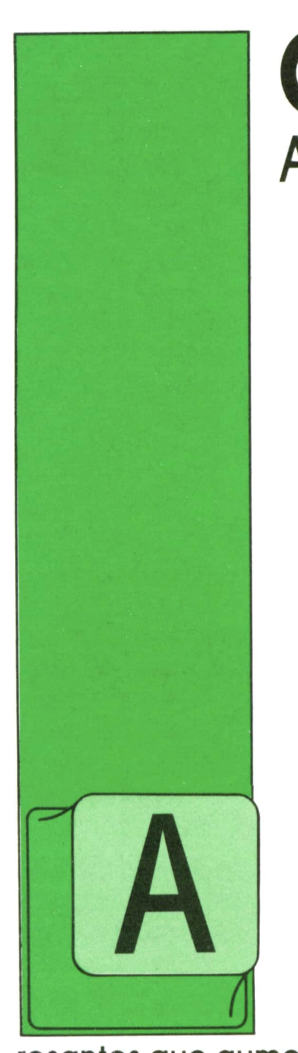
 while not eof (Fichero) do
   (* mientras quede algo, presentar líneas: *)
   begin
     while not eoln (Fichero) do
       (* mientras queden caracteres en la línea actual: *)
       begin
        read (Fichero,C);                     (* leer carácter *)
        write (C)                             (* presentarlo    *)
       end;
     readln (Fichero);  (* tras cada línea, pasar a la siguiente *)
     writeln
   end;

 close (Fichero)
end.
```

Realmente, con la mayoría de los compiladores podríamos haber leído cada línea con una sola operación:

```
while not eof (Fichero) do
  begin
    readln (Fichero,Renglon);
    writeln (Renglon)
  end;
```

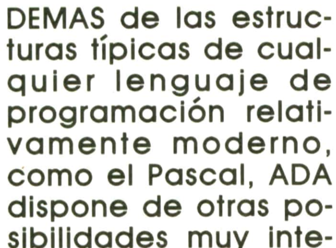
Renglon sería una variable como la del programa Guardar.



OTROS LENGUAJES
ADA



Sintaxis del lenguaje



DEMAS de las estructuras típicas de cualquier lenguaje de programación relativamente moderno, como el Pascal, ADA dispone de otras posibilidades muy interesantes que aumentan su flexibilidad de cara a la programación (aunque algunos hayan objetado que hacen demasiado extenso el lenguaje para su aprendizaje).

Hablemos de estas diferencias que hacen del ADA un lenguaje tan potente.

— Los atributos denotan ciertas características predefinidas de los nombres a los que se refieren (éstos pueden ser variables, constantes, etc.).

Por ejemplo, la expresión:

CARD'ADDRESS

se refiere a la dirección (address) de la variable CARD, que puede ser un registro. Esto permitiría utilizar la dirección de CARD en expresiones tipo ADDRESS directamente.

— Los agregados (aggregate) son valores de variables compuestas, como matrices o registros manejados sin la necesidad de selectores (imprescindibles en PASCAL, por ejemplo, para referirse a componentes de un registro).

Esto permite el manejo de datos de tipo compuesto sin la necesidad de sentencia WITH, lo que hace posible, por ejemplo, la asignación de valores a un registro o matriz, directamente.

— ADA permite las conversiones de tipo para evitar la enorme dificultad de los tipos estrictos (al igual que en MODULA-2).

— Una expresión calificadora *(qualified expressions)* se utiliza para dar expresamente el tipo de la expresión o agregado a utilizar. Esto permite la detección de errores si éste se asigna a una variable de otro tipo.

— La ejecución de un localizador (allocator) crea un objeto y proporciona como resultado un valor accesible que designa al objeto.

— Una expresion estática se define en término de sus posibles constituyentes y debe ser calculable en el momento de la compilación, tras la cual quedará inalterable.

— Un bloque (block) es un conjunto de sentencias agrupadas con un identificador, al estilo de los procedimientos de PASCAL (PROCEDURES), que pueden tener también parte declarativa y que son ejecutables y compilables de forma independiente.

— El «overloading» o sobrecarga es una situación permitida en ADA. Esta se produce cuando se nombra a dos subprogramas u operadores diferentes con el mismo identificador. El compilador discrimina a cuál nos referimos gracias a los tipos de datos que se utilicen como

parámetros. Estos deben ser diferentes para cada versión de subprograma u operador que utilicemos.

Obsérvense las enormes posibilidades de esta situación. Podríamos utilizar, por ejemplo, el símbolo * (que normalmente denota multiplicación) para realizar otras operaciones sobre datos en las que no estuviese definida, como multiplicar matrices, combinar registros, etc.

Por ejemplo:

function "*" (X,Y:MATRIX) return MATRIX

Seguido de la definición y el cuerpo de la función, como en PASCAL, definiría un producto de matrices, con el cual sería posible la expresión:

A=B*C

Siendo todas las variables matrices.

— Una estructura esencial en ADA, responsable de la modularidad, son los paquetes (Package). Estos son similares a los módulos de MODULA-2.

— Los tipos privados de datos, utilizados dentro de un bloque, pero accesibles desde el exterior, son tipos cuyas características son desconocidas por el programa exterior, pero que se puede acceder a ellos si son asignados a otras variables que puedan contenerlos.

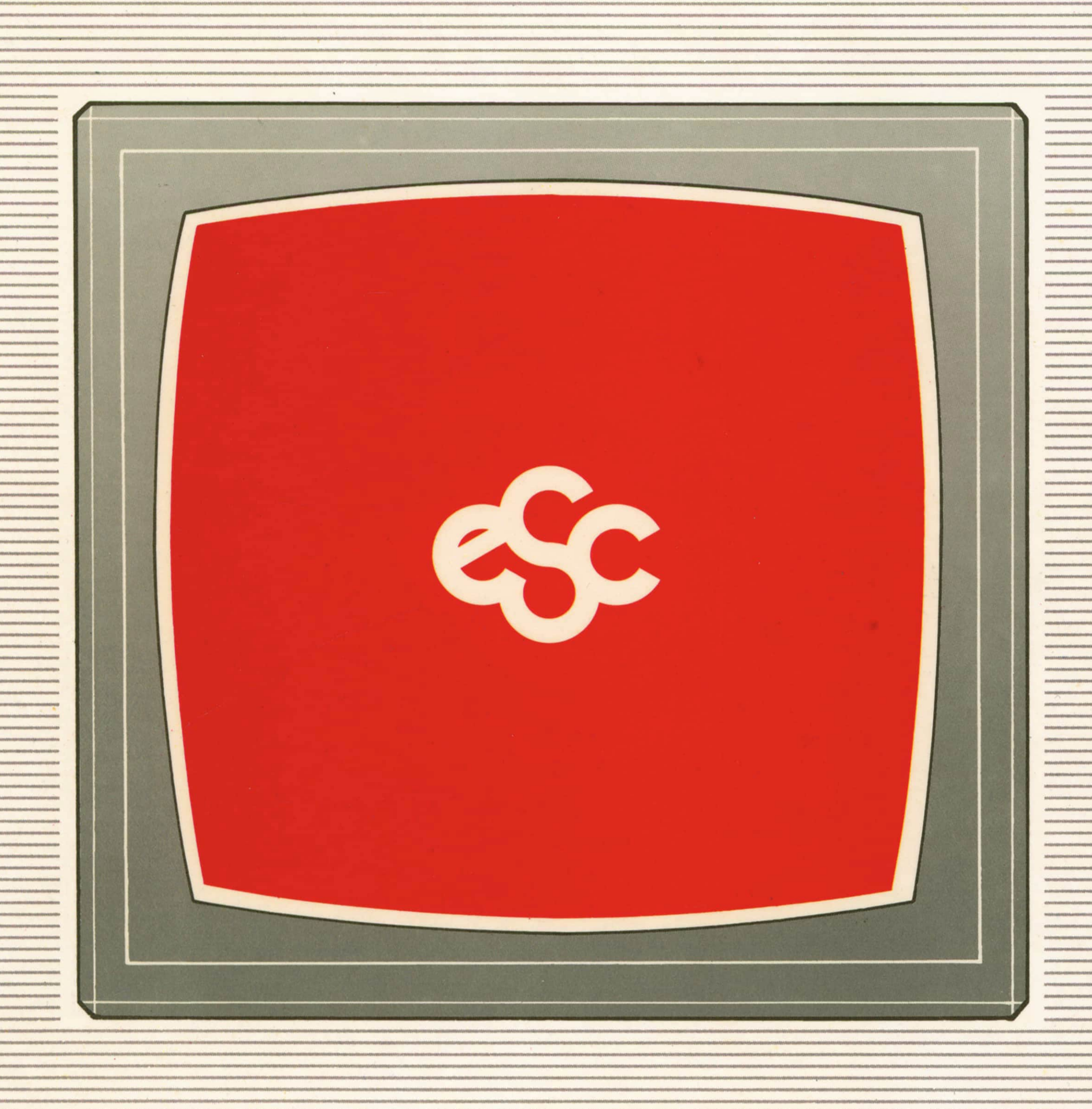
— La flexibilidad proporcionada por los bloques de definición hace posible en ADA utilizar bloques implementados en otros lenguajes, con la restricción de que el bloque de definición esté en ADA.

— Las cláusulas de uso *(use)* permiten utilizar las variables visibles internas a un paquete, en la zona del programa donde se ejecuten dichas cláusulas. Esta posibilidad permite una mayor flexibilidad, pero es peligrosa, ya que saca de contexto variables, posibilitando problemas de dominio.

— Las declaraciones de cambio de nombre *(renames)* permiten cambiar el identificador de cualquier objeto lógico. (Por ejemplo, el nombre de una variable, etcétera), en tiempo de ejecución.

— En un bloque, todos los tipos de datos predefinidos tienen un entorno estándar si el nombre del bloque va precedido por la palabra STANDARD. En caso contrario, podremos definir libremente nuestras propias operaciones, etc.

En el siguiente capítulo seguiremos hablando de las principales diferencias entre el lenguaje ADA y los demás lenguajes conocidos.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼